1-1-1988

Prototyping management information systems application software : an empirical study of developers' perceptions.

David Lawrence Russell
University of Massachusetts Amherst

PROTOTYPING MANAGEMENT INFORMATION SYSTEMS APPLICATION SOFTWARE: AN EMPIRICAL STUDY OF DEVELOPERS' PERCEPTIONS

A Dissertation Presented

by

DAVID LAWRENCE RUSSELL

Submitted to the Graduate School of the
University of Massachusetts in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 1988

School of Management

PROTOTYPING MANAGEMENT INFORMATION SYSTEMS APPLICATION SOFTWARE: AN EMPIRICAL STUDY OF DEVELOPERS' PERCEPTIONS

A Dissertation Presented

by

DAVID LAWRENCE RUSSELL

Approved as to style and content:

Van Court Hare, Jr., Chairperson of Committee

Craig L. Moore, Member

David W. Stemple, Member

Conrad A. Wogrin, Member

D. Anthony Butterfield,
Ph.D. Program Director
School of Management

For Nancy,

Jimmy, Ellen and Jono,

for whom it was done

ACKNOWLEDGMENTS

How can I begin to thank the myriad of people and organizations who have helped me achieve the objective of earning the Ph.D. degree? If I list them, there is the very grave risk of inadvertently leaving off a name. And, if I do name them, does the order in which I thank them imply their degree of importance? I owe so many thanks to so many people, that I truly do not know where to begin.

My wife, Nancy, is one of the two central figures in this process. Soon after we were married in 1978, I returned to night school to earn an M.S., and I've been going to school ever since. She has kept the home fires burning ever since, and given me three wonderful children along the way. I hope we can now have a normal life together.

My Committee Chair and academic advisor, Dr. Van Court Hare, Jr., is the other central figure. Van has guided my progress throughout my time at UMass. His paternal (in the best sense of the word) caring serves as a model for me as I pursue my own academic career. My Committee members, Craig Moore, Dave Stemple and Connie Wogrin have helped me more than they'll ever know.

My employer, Western New England College, has supported my graduate work both monetarily and otherwise. My thanks go to Dean Stan Kowalski and my Department Chairs, Drs. Ginny Knight, Ed Sandifer and Marilyn Pelosi, who each in

their turn have smoothed the road for me. My office mate, Dr. Jerzy Letkowski, assisted me with some of the illustrations and my Department's secretary, Mrs. Pat Shanley, was an island of sanity and stability in a world that was sometimes quite the opposite.

Several good people have helped me with the editing of this dissertation. My fellow Ph.D. candidates in MIS, Carl Chimi and Marie Wright, have been good friends and valued colleagues (I've heard us referred to as "the three musketeers" on the third floor of SOM), and I thank them for their work on this manuscript. My father, Larry Russell, also carefully edited this manuscript despite recovering from a grave illness.

I must also take the time to thank the respondents who are the basis of this research. They freely gave of their time and expressed much interest in my research.

In a work of this scope, flaws inevitably remain. I am solely responsible for these.

ABSTRACT

PROTOTYPING MANAGEMENT INFORMATION SYSTEMS APPLICATION SOFTWARE: AN EMPIRICAL STUDY OF DEVELOPERS' PERCEPTIONS

SEPTEMBER 1988

DAVID L. RUSSELL, B.A., UNIVERSITY OF WISCONSIN

M.S.L.S., UNIVERSITY OF WISCONSIN

M.S., RENSSELAER POLYTECHNIC INSTITUTE

Ph.D., UNIVERSITY OF MASSACHUSETTS

Directed by: Professor Van Court Hare, Jr.

Information systems often are delivered which fail to meet the user's expectations. One appropriate response is user-involvement in systems design; in recent years, a particular form of user-involvement, prototyping, has gained popularity. When prototyping, the designer and user work together to build the system iteratively. Earlier empirical investigations indicate that the prototyping environment fosters user acceptance and user satisfaction. There are also some indications that prototyped systems require fewer programmer-hours of effort, and thus can be delivered at less cost.

This dissertation addresses an heretofore unresearched aspect of prototyping: the developer's perspective. Specif-

ically, we investigate what motivates developers to choose prototyping. Although there is a broad consensus of opinion regarding what prototyping is, our findings indicate that prototypers, as a group, do not believe that prototyping produces systems in significantly less time or at significantly less cost than systems developed in the conventional manner. However, developers do perceive that prototyped systems are of significantly greater quality than conventionally-developed systems.

These findings give rise to considerable discussion, in which we interpret that there exist other motivations for prototyping. A more collegial relationship with clients, a reduction in anxiety when developing systems and a greater degree of satisfaction are among the factors motivating the choice to prototype. In addition, we find there exists an indirect economic incentive to prototype, based on the reuse of prototyped modules in subsequent development efforts.

TABLE OF CONTENTS

ABSTRACT.....vii

LIST OF FIGURES.....xii

LIST OF TABLES.....xiii

1 INTRODUCTION.....1

1.1 Systems Development Life Cycle ("SDLC").....2
1.2 Problems with the SDLC.....8
1.3 User-involvement in the SDLC.....17
1.4 Prototyping.....20
1.5 Scope of this dissertation.....21

2 DEFINITION, APPLICATION AND THEORETICAL BASIS.....23

2.1 Definition.....23
2.2 Prototyping in other fields.....25
2.3 Early prototyping concepts in MIS literature.....26
2.4 Applications of prototyping.....35

2.4.1 "Rapid" vs. "conventional" prototyping.....35
2.4.2 Modeling vs. evolving.....37
2.4.3 Design vs. development tool.....40

2.5 Alternatives to prototyping.....42
2.6 Theoretical basis of prototyping.....43

2.6.1 Applicable theoretical models.....45
2.6.2 Prototyping in MIS research.....48
2.6.3 Prototyping in scientific research.....50

2.7 Conclusion.....51

3 PROTOTYPING: EMPIRICAL INVESTIGATIONS.....53

3.1 Alavi and Henderson, 1981.....53
3.2 Boehm, et al., 1984.....54
3.3 Alavi, 1984.....56
3.4 Mahmood, 1987.....58
3.5 Necco, et al., 1987.....60
3.6 Conclusion.....61

4 RESEARCH HYPOTHESES AND RESEARCH DESIGN.....63

4.1 Research hypotheses.....63
4.2 Research design.....66

4.2.1 Respondent qualifications.....67
4.2.2 Sample development.....67
4.2.3 Evaluation of the interview.....70
4.2.4 Sample size determination.....71

5 RESEARCH FINDINGS.....72

5.1 Demographic analysis.....72
5.2 Research findings.....77

5.2.1 Commonness of definition.....77
5.2.2 Time dimension.....83
5.2.3 Cost dimension.....94
5.2.4 Quality dimension.....102
5.2.5 Conclusion.....110

5.3 Discussion.....111

5.3.1 Direct economic motivators.....112
5.3.2 Other motivators.....116
5.3.3 Interpretation.....127

6 CONCLUSION.....134

6.1 Reiteration of research findings.....134
6.2 Implications for systems development actors.....136

6.2.1 Systems developers.....136
6.2.2 Clients.....139
6.2.3 Educators.....140

6.3 Limitations.....141
6.4 Future research.....143

6.4.1 Research generated by this dissertation.....144
6.4.2 Other research issues.....145

EXHIBITS

EXHIBIT A: Interview instrument.....148
EXHIBIT B: Interview evaluation instrument.....157
EXHIBIT C: Transcript of Responses to Selected Interview Questions, Respondent 1.....164
EXHIBIT D: Transcript of Responses to Selected Interview Questions, Respondent 2.....170
EXHIBIT E: Transcript of Responses to Selected Interview Questions, Respondent 3.....176
EXHIBIT F: Transcript of Responses to Selected Interview Questions, Respondent 4.....181
EXHIBIT G: Transcript of Responses to Selected Interview Questions, Respondent 5.....187
EXHIBIT H: Transcript of Responses to Selected Interview Questions, Respondent 6.....194
EXHIBIT I: Transcript of Responses to Selected Interview Questions, Respondent 7.....202

EXHIBIT J: Transcript of Responses to Selected Interview Questions, Respondent 8.....207
EXHIBIT K: Transcript of Responses to Selected Interview Questions, Respondent 9.....216
EXHIBIT L: Transcript of Responses to Selected Interview Questions, Respondent 10.....221
EXHIBIT M: Transcript of Responses to Selected Interview Questions, Respondent 11.....226
EXHIBIT N: Transcript of Responses to Selected Interview Questions, Respondent 12.....232
EXHIBIT O: Transcript of Responses to Selected Interview Questions, Respondent 13.....236
EXHIBIT P: Transcript of Responses to Selected Interview Questions, Respondent 14.....241
EXHIBIT Q: Transcript of Responses to Selected Interview Questions, Respondent 15.....247
EXHIBIT R: Transcript of Responses to Selected Interview Questions, Respondent 16.....253
EXHIBIT S: Transcript of Responses to Selected Interview Questions, Respondent 17.....258
EXHIBIT T: Transcript of Responses to Selected Interview Questions, Respondent 18.....265
EXHIBIT U: Transcript of Responses to Selected Interview Questions, Respondent 19.....271
EXHIBIT V: Transcript of Responses to Selected Interview Questions, Respondent 20.....275
EXHIBIT W: Transcript of Responses to Selected Interview Questions, Respondent 21.....280
EXHIBIT X: Transcript of Responses to Selected Interview Questions, Respondent 22.....287
EXHIBIT Y: Transcript of Responses to Selected Interview Questions, Respondent 23.....291
EXHIBIT Z: Transcript of Responses to Selected Interview Questions, Respondent 24.....296
EXHIBIT AA: Transcript of Responses to Selected Interview Questions, Respondent 25.....302
EXHIBIT AB: Transcript of Responses to Selected Interview Questions, Respondent 26.....307
EXHIBIT AC: Transcript of Responses to Selected Interview Questions, Respondent 27.....315
EXHIBIT AD: Transcript of Responses to Selected Interview Questions, Respondent 28.....322
EXHIBIT AE: Transcript of Responses to Selected Interview Questions, Respondent 29.....331

BIBLLIOGRAPHY.....335

LIST OF FIGURES

Figure 1.1: Idealized systems development.....3
Figure 1.2: Popular perception of communications difficulties in systems development.....12
Figure 1.3: Conceptual versus Actual systems development.15

Figure 2.1: Learning Cycle Model.....47
Figure 2.2: Comparison of Lewin-Schein and Kolb-Frohman models.....49

Figure 5.1: Respondents by type of employment.....78
Figure 5.2: Perceived career impact of prototyping.....84
Figure 5.3: Time dimension.....86
Figure 5.4: Specific project time perception.....89
Figure 5.5: Cost dimension.....96
Figure 5.6: Specific project cost perception.....98
Figure 5.7: Quality dimension.....104
Figure 5.8: Financial impact of prototyping.....114
Figure 5.9: Degree of anxiety when prototyping.....118
Figure 5.10: Perception of leadership role.....121
Figure 5.11: Presentation of systems options.....124

LIST OF TABLES

Table 5.1: Demographic analysis.....75
Table 5.2: Cross-tabulations of respondent employment with time, cost and quality dimensions.....92
Table 5.3: Cross-tabulation of time and cost dimensions..101
Table 5.4: Cross-tabulation of quality dimension with time and cost dimensions.....107

CHAPTER 1

INTRODUCTION

That software development faces a crisis today is well-known. Systems very often are delivered late; more seriously, systems are delivered which fail to meet the user's expectations with regard to functionality, appearance, and other important dimensions. In recent years, it has become apparent that at least some of the blame for this state of affairs is inherent in the conventional way systems are developed. Convention holds that one must fully and completely understand user requirements and specifications before software development can take place. Thus, the conventional approach views systems analysis and development as, to quote a popular text, "an orderly, structured process for identifying and solving problems" [Gore and Stubbe, 1983, p. 7].

Systems analysis and design, in the sense of an orderly, sequential process for developing computer systems, has a long-standing intellectual heritage. In the 1960's and early 1970's, several substantial texts addressed the topic [Hare, 1967; McMillan and Gonzalez, 1973]. This heritage will be discussed briefly here as a means of giving historical context to the discussion that follows.

The word "systems" in the term "systems analysis" seems to have derived from the general notion of a system, and not the restricted meaning of the term "computer system". Early work focused on the development of mathematical models to describe some system in process. Hare [Hare, 1967], for example, expended considerable effort on the process of "simplification", that is, applying the scientific principle of parsimony to the multi-faceted phenomena under study. Once the underlying principles are found, he argued, one can proceed to model the system under study. He paid particular attention to simulation tools to achieve this modeling. McMillan and Gonzalez [McMillan and Gonzalez, 1973] concentrated rigorously on the simulation approach.

1.1 Systems Development Life Cycle ("SDLC")

Conceptually, the system realization process is a smooth progression in which: (1) the designer formulates a design concept; (2) the designer presents the design to a decision maker, who approves the design; (3) the designer implements the system for the client; followed by (4) implementation of the system. Figure 1.1, after Swanson [Swanson, 1988, p. 35] encapsulates this concept. It is clear that, in practice, this progression is fraught with difficulty.

A major step forward occurred in the 1970's. As will be explained shortly, Boehm [Boehm, 1973] and Davis [Davis, 1974], apparently independently, evolved the concept of the systems developed life cycle ("SDLC")[1] in 1973. Since that time, the SDLC has become the major integrating theme of

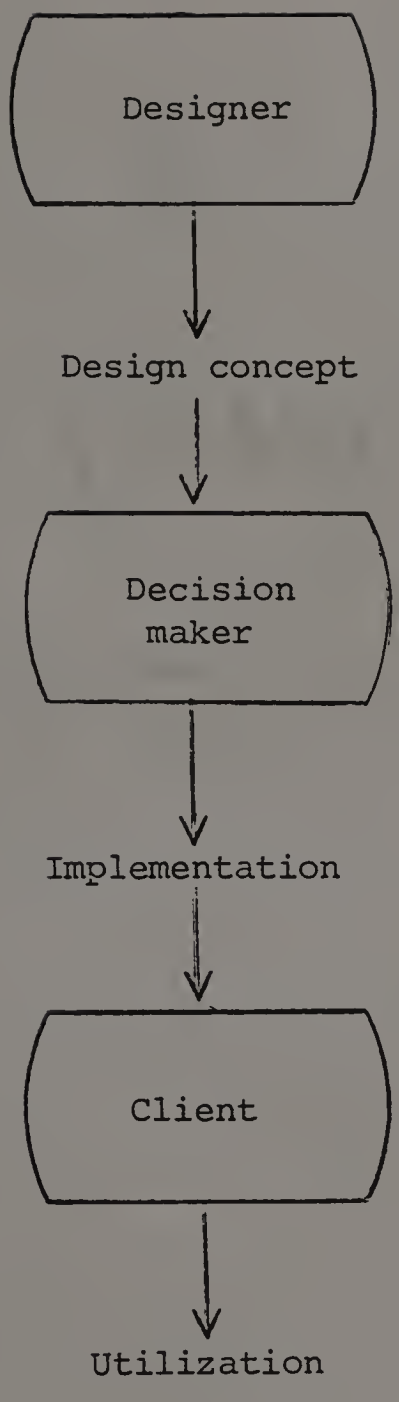


Figure 1.1

Idealized systems development

(after Swanson, 1988, p. 35)

systems development. It states that the development of a system proceeds, or at least should proceed, in an orderly, prescribed fashion. The following life cycle model, an amalgam taken from Davis and Olson [Davis and Olson, 1985, pp. 570-577], Murdick [Murdick, 1980] and the author's own perceptions, is typical.

Planning stage: The goals and objectives of the prospective system are established. An initial feasibility study is generally called for.

Definition stage: The system is defined in terms of what it shall and shall not include. The goals and objectives of the system are established. Following a feasibility assessment, initial information is gathered. A high-level conceptual design completes this stage.

Analysis stage: Analysis of the existing system leads to a greater understanding of the problem domain and the specific tasks to be undertaken by the proposed system. A more detailed feasibility study may be called for, together with a major financial commitment from user management.

Design stage: The understanding gained in the previous stages results in the physical and logical design of the proposed system. The result of this phase is a detailed specification document, which, de facto if not de jure, establishes a contract between the developers and the recipients of the system.

Coding: The system is coded in an agreed-upon language according to the specifications detailed in the previous stage.

Debugging: As each component is developed, it is test-run and the resulting errors are corrected. As will be seen in the discussion below, the errors found and corrected at this stage are generally of a syntactic nature.

Implementation: The system developed in the previous stage is placed into service. Typically, a training period takes place in which users are oriented to the new system and gain the skills needed to use it.

Cutover: Using any one of variety of techniques, the old system ceases and the new system begins. Data conversion may be required.

Maintenance: Remaining bugs, consisting primarily of syntax errors, are fixed ("corrective maintenance") but it is known that a good deal of effort in this stage is focused on altering the characters of the new system to conform to user preferences ("perfective maintenance"), that is, semantic errors. [Guimaraes, 1983]. In fact, Lientz and Swanson [Lientz and Swanson, 1980B] claim in an exhaustive survey that more than 50% of all maintenance can be considered perfective, while nearly another 24% is "adaptive", which in part addresses changes in input data structures. This is further discussed by Swanson [Swanson, 1988].

In recent years, it has become obvious that the final stage, maintenance, consumes an ever-increasing portion of the total cost of a computer system. Ironically, in successful systems that are used a good deal, this last stage is particular demanding of total system cost [Boehm, 1973]. This seeming paradox can be explained if one understands that maintenance consists of more than fixing "bugs". Most maintenance is perfective, that is, adapting the system to the user's evolving needs, rather than corrective [Guimaraes, 1983; Swanson, 1988]. It follows that successful systems, on which users depend, demand continual updating.

Two additional stages should be considered. One stage addressed by few authors is the death of the system [Henderson and Ingraham, 1982]. This author hypothesizes that the lack of explication of this stage results in many systems being "patched" when they should be replaced. This topic is not the subject of this dissertation, however.

A stage which seldom occurs is an active process of reviewing the system once implemented. Called the "post-audit" stage by Davis and Olson [Davis and Olson, 1985, p. 571], this activity is intended as a learning and feedback process for the systems developer. This topic is also not the subject of this dissertation.

The near-universal acceptance of the SDLC model can be seen in many aspects of the information systems discipline. Besides commonly serving as an integrating outline in information systems texts, the SDLC also serves as a unifying

theme for discussions of systems development. For example, in 1980, a major conference on systems analysis and design chose the SDLC as its organizing model [Cotterman, et al., 1981], though not without dissent [McCracken and Jackson, 1981].

Despite the great number of enhancements to the basic SDLC model that have been proposed, all have a common starting sequence. The system developer, whether an individual or a team, must:

1) understand the existing system ("systems analysis");

2) understand what the user requires in a proposed system ("systems specification"); and

3) design a system in response to user needs before undertaking actual development ("systems design").

The SDLC model evolved in the early 1970's. In the field of software engineering, it was mentioned by Boehm in his oft-cited 1973 Datamation article [Boehm, 1973] but was articulated more clearly by him several years later [Boehm, 1976]. The model was developed separately, and apparently independently, by Davis in the field of information systems [Davis, 1974, pp. 413-420]. Davis, however, limits his model to the development of systems per se, while Boehm and others extend the model to the entirety of the systems analysis, design and development process. It is clear that the SDLC model has permeated much of the thinking in the field

of software development for many years before its articulation by Boehm and Davis [see, for instance, Ackoff, 1967]. Indeed, the model fulfills Harel's [Harel, 1980] requirements for being a "folk theorem".

The SDLC model marks a major break from the mathematically- and simulation-oriented concept of systems analysis described earlier. In the works cited earlier [Hare, 1967; McMillan and Gonzalez, 1973] there is no indication that this model was known. Indeed, the SDLC constitutes a significant management tool for the development of information systems.

While the SDLC is commonly presented as sequential, in fact it is a feedback-driven process. Various authors have dealt with the feedback loops in different ways. Davis [Davis, 1974; Davis and Olson, 1985] describes specific feedback iterations. Others [e.g., Murdick, 1980] generally discuss the concept of feedback and iteration without describing specific iteration paths. Without exception, however, all clearly call for the analysis and specification stages to precede the design stage, which in turn is to precede the development and implementation stages.

1.2 Problems with the SDLC

The fundamental need for analysis and design prior to software development has been challenged by many authorities. The best-known challenge is Russell Ackoff's "Management Misinformation Systems" [Ackoff, 1967]. The most-quoted passage in this most-quoted piece of the infor-

mation systems literature succinctly stated the futility of the traditional approach:

> For the manager to know what information he needs he must be aware of each type of decision he should make (as well as does) and he must have an adequate model of each. These conditions are seldom satisfied. [Ackoff, 1967, p. B-149]

Ackoff was not alone. Another oft-cited piece, Powers and Dickson's "MisProject [sic] Management: Myths, Opinions and Reality" [Powers and Dickson, 1973], noted that in practice systems often exist with vaguely specified objectives. In doing so, the authors discredited the myth that objectives must be established clearly and definitively prior to software development, noting that "the clarity of project objectives was not related to user satisfaction" [Powers and Dickson, 1973, p. 153]. Davis later continued this line of reasoning [Davis, 1982, p. 19].

In large measure, the SDLC model is doomed to fail even if all steps in it are performed flawlessly. This is because up to 70% of the total system development effort is invested before the user actually sees what s/he has ordained [Henderson and Ingraham, 1982]. Between the original motivation for the system and the actual implementation of the system come a wide variety of decisions, assumptions and default values that color and characterize the resulting system.

In a typical scenario, a substantial amount of time passes between the initial motivation for the system and the actual delivery of the system. Yet, systems are developed in accordance with the needs of the user at the time the

system is called for. The developer separates him/herself from the user, with the possible exception of occasional "check backs" to verify information or formally report progress. It is assumed that the developer has a clear understanding of the problem prior to actual development of the system, and that the user's needs will remain static during the development period.

A further assumption is made: that the user is capable of clearly and succinctly expressing his/her needs. As the developer proceeds with the design of the system, further interactions with users tend to be in the nature of fact verification, rather than in-depth consultation. This is most unfortunate, since many design decisions affect the resulting system. For example, the type of file access provided is often chosen in the design stage. Yet, the decision between sequential and random access dramatically affects how the user will deal with the system.

Developers defend not dealing with users on such technical topics by noting that users are typically ignorant of computing topics, and the developer's decision is likely to be correct. What is unfortunate is the lack of a readily-available medium of communication between the user and the developer on this and other system issues. As Andrews notes, "the main source of ... errors lies in the inability of the systems analyst to translate [the user's] wants and needs into a functioning system" [Andrews, 1983, p. 17].

This inability to translate user needs into a workable information system should not surprise us. It goes beyond the seeming inability of users to express their needs and beyond system decisions that impact the user. The problem goes to the heart of human intelligence and decision-making. Many subject experts, even when they understand their needs completely, are quite incapable of expressing the processes by which they make decisions or exercise expertise. The problem can be summarized as follows: the developer must develop a system that will be executed on a distinctly unintelligent computer. Yet, s/he does not have access to the underlying needs and processes to be addressed by the system, as they are clouded by the processes of human intelligence. Thus, a communications problem exists. It is no surprise that it is nearly impossible to clearly elucidate user needs and processes clearly enough to achieve a correct system from the data derived from analysis and specification of the system. Figure 1.2, taken here from Simkin [Simkin, 1987, p. 7], is a popular cartoon expressing the communications difficulties inherent in systems development.

The traditional development methods have generated what Wetherbe and Berrisford term an "expectations gap" [Wetherbe and Berrisford, 1979, p. 10; see also Wetherbe, 1984]. Users expect that the system they want will serve their needs. They often express surprise and resentment when, often months after they first met with the developer, the delivered system strikes them as a bizarre convolution of their expressed needs. Given the difficulties of communica-

Figure 1.2
Popular perception of communications difficulties in systems development
(after Simkin, 1987, p. 7)

tion present, difficulties which can be exacerbated by the SDLC, what is surprising is that successful systems exist at all.

It is important to note that users and developers need not experience the frustration of unsatisfactory systems. Caring, sensitive developers exist, as do open, articulate and receptive users. Managements that are aware of the dynamics involved in systems development also exist. The major point here is: there is nothing in the traditional systems development life cycle model that will avoid these communications difficulties, and the feelings of frustration that derive.

The present author posits that a lack of communication with users in the eliciting of user needs is at the core of the systems development problem. This has been empirically demonstrated. Jenkins et al. note, for example, that traditional development methods resulted in users claiming that major reporting requirements were missed in 65% of the cases studied [Jenkins, et al., 1984].

Practitioners bear the brunt of the difficulties discussed here. One alternative would be to consciously backtrack over previous stages of development, taking advantage of new insights and understandings as they arise. This approach, however, implies ongoing communication between developer and user as the system is built. This seldom takes place in practice. Part of the reason is that "conventional systems development practitioners view such recycling as bad practice" [Young, 1984, p. 154].

In reality, the SDLC needs to be a highly iterative process. Instead of a steady progressive slope that is often pictured from beginning to end, the actual curve is a slowly looping spiral (figure 1.3). Such a curve epitomizes the iterative nature of systems development, and it is clear that the developer is expected to have a clear understanding of the user's needs prior to entering the later stages of the life cycle. What the life cycle model does is to delineate clearly the definition, analysis, development and implementation stages.

As a result of separating the development stages, a substantial time lag exists between the time the user initiates development and the time the system is delivered. This time lag is at the root of many systems development problems. Since information systems model procedures and activities in the user's environment, the user's needs change at least as often as does the user's environment. Thus, even if the definition, analysis and design stages were performed flawlessly, the resulting system would necessarily fail to meet the user's needs as those needs evolved during development. Further, the presence of an automated system itself changes the environment; of course, the system was designed to address the environment <u>prior to</u> the system-induced changes [Guimaraes, 1985]. And, even if a system were delivered in a flawless state, it would soon fail to meet the needs of users. This is because all systems are in

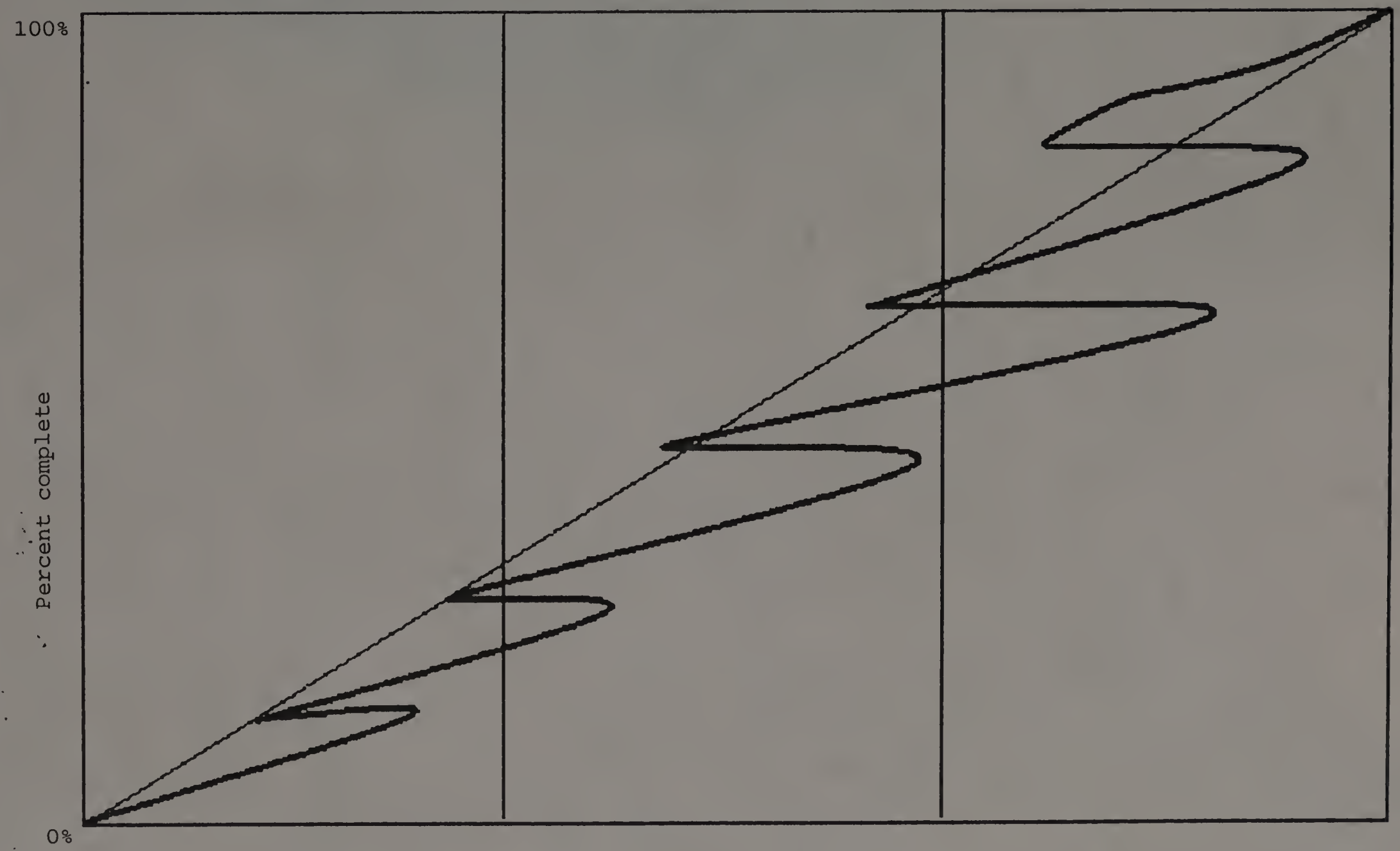


Figure 1.3

Conceptual versus Actual systems development

a state of continuous evolution, and must continuously change to deal with the world they model [De and Hse, 1985].

The crucial nature of the development stage has been highlighted well by Ginzberg [Ginzberg, 1981]. He claims that the likelihood of success can be estimated at this stage. Great savings can be achieved if systems destined to fail could be aborted at this stage. He offers guidelines for identifying likely failures.

Some methodology is clearly needed to avoid the communication problems associated with the early development stages. This methodology would assist developers and users to communicate effectively. Indeed, developers have adopted a wide variety of techniques to deal with this problem and the subsequent problem of conveying the specifications derived to the programmer. Over time, a wide repertoire of analysis and design tools evolved. The primary purpose of many was to facilitate communication between the developer and the user. Cougar provides a useful and comprehensive summary of the technologies of the 1960's and early 1970's in his overview article, "Evolution of Business Systems Analysis Techniques" [Cougar, 1973]. Davis' discussion is also useful [Davis, 1982, pp. 14-19].

There has been no shortage of discussion regarding these problems. Waters feels that existing analysis tools fail since they are not comprehensive enough to present a realistic model of the proposed system, particularly regarding system details [Waters, 1979]. Lientz and Swanson discuss a number of productivity aids, primarily with a view to

their impact on maintenance. Despite all the productivity aids available, they conclude that "the findings point clearly to the importance of the relationship established with the users of the application systems... [it is] suggested that new aids be directed, in part, specifically to the user interface" [Lientz and Swanson, 1980B, p. 120].

Toward this end, a wide variety of alternative methods are available for development, but, as King notes, organizational factors often inhibit their use [King, 1982]. One class of these alternative design methods is structured analysis, which was popular in the late 1970's and early 1980's [DeMarco, 1982; Gane and Sarson, 1979; Yourdon, 1975]. Yet, at best these techniques have proved disappointing. As Yourdon notes [Yourdon, 1986], these techniques were laborious and took a static view of systems, just as systems were becoming ever more complex, interactive and existing in real time.

1.3 User-involvement with the SDLC

A near-universal response to these problems has been "user-involvement". A simple definition of user-involvement is the participation in the development process by users of the intended system [Olson and Ives, 1984]. It is argued that user-involvement is the key to successful system implementation [Holmes, 1978; King and Rodriguez, 1978; Kling, 1977; Swanson, 1974]. Maish claims that positive feelings by users toward an MIS is correlated with user-involvement, while user behavior toward the MIS is not significantly

correlated with user-involvement [Maish, 1979]. These arguments, however, are much in dispute. Academics [Ives and Olson, 1984; Olson and Ives, 1984] claim there is no empirical basis for the assumption. Practitioners early on noted how difficult it is to have users, particularly managers, be involved with the development of their systems [e.g., Davis and Taylor, 1975; Gibson, 1977; Kneitel, 1977].

Nonetheless, academic sources have produced a wide variety of models and techniques to increase user-involvement [e.g., Swanson, 1974]. King and Cleland advocated the establishment of interpersonal models to guide and manage developer-user interaction [King and Cleland, 1971]. Others take the perspective of internal politics and organizational behavior [Keen and Gerson, 1977; Kling and Iacono, 1984; Markus, 1983].

User-involvement is easier to call for than to practice. Dagwell and Weber's research has shown that users and developers often operate with different mindsets. They point out that developers have a "Theory X" view of users, particularly for clerically-oriented data processing systems. Designers take on a limited, collegial "Theory Y" view when developing professionally-oriented management information systems [Dagwell and Weber, 1983]. Their work in turn is ground in the research of Hedberg and Mumford, who focus on the values (social and political, as well as technical) and behaviors that developers bring to systems. Hedberg and Mumford concluded that "the most important thing seems to be to establish real communication between experts

and clients in design groups and to change the content of the joint discussion" away from purely technical matters [Hedberg and Mumford, 1975, p. 58]. Much attention in the academic press has focused on ways to improve system designers' attitudes and thus increase user involvement.

Also based on this line of research is Salaway's findings on the relationship of the developer's behavior on systems development. Advocating behavior modification techniques, she argues that developer behavior and the attitudes the behavior displays can greatly aid or harm the development process [Salaway, 1987].

A claimed link between user-involvement and systems success provides the linchpin for this argument (a claim disputed by Ives and Olson [Ives and Olson, 1984]; see above). Edström, for example, posits communication models to facilitate user involvement, noting that the "practical importance of user influence ranges well beyond the development of MIS to almost all areas of design activities ... [it is] important to move beyond the simple relationship between user influence and the perceived success of the system" [Edström, 1977, p. 606].

DeSanctis and Courtney have followed the same reasoning, but have developed their ideas from an organizational development perspective. The argue for a bridge between organizational development techniques and information systems. They especially note that cognitive and emotional information must be passed from user to developer, but that

traditional tools focus solely on the passage of technical information [DeSanctis and Courtney, 1983].

1.4 Prototyping

"Getting it right the first time" is a common theme in information systems literature. The strategic benefit of quickly developing a correctly functioning information system has been established [Ives and Learmonth, 1984]. Many tools are intended to accomplish this objective, most of which focus on user-involvement. Within the domain of user-involvement, there is one tool of particular promise: the prototyping method of systems development, or, more simply, "prototyping".

In the past ten years, prototyping has received much attention as an alternative development mode. Using prototyping, the user and the developer have an ongoing, symbiotic relationship as the system evolves [Naumann and Jenkins, 1982]. Although hinted at since the 1960's [Ackoff, 1967], this method of systems development did not exist prior to the mid-1970's because there did not exist appropriate tools to implement it [Sarvari, 1983]. As software has taken an increasingly larger percentage of information systems' budgets [Boehm, 1973], however, the costs of software failure have focused attention on delivering systems that are not only technically correct, but conform to the user's expectations for the system as well. Prototyping is intended to foster such a collegial relationship between user and developer from the time the system is initiated until it is finally delivered [Naumann and Jenkins, 1982]. There are

indications that prototyping is increasingly being used. The percentage of firms reported as using prototyping increased from 4% in 1984 [Jenkins, et al., 1984] to 45% in 1987 [Necco, et al., 1987]. Necco et al. further reported that prototyping was primarily used for on-line transaction processing systems and ad hoc reporting tasks [Necco, et al., 1987, p. 470].

1.5 Scope of this dissertation

This dissertation will address the relationship between developers and users when the prototyping method of systems development is employed, with strong emphasis on the developer's perceptions. It will concentrate on the development of application software of the type generally associated with Management Information Systems ("MIS"). The prototyping of systems using database management systems ("DBMS") on micro- and small mini-computers will be emphasized. Within this class of development software, ancillary tools are heavily emphasized.

In the next chapter, we will examine the historical motivation for prototyping, its benefits and drawbacks, its appropriate uses and its potential. The discussion will be limited to application programs typical of management information systems. Nonetheless, it is anticipated that the findings of this dissertation will be applicable to other areas of software development.

Notes

1. The acronym "SDLC" should not be confused with an identical acronym used in the telecommunications field. That acronym stands for "Synchronous Data Link Control", a proprietary IBM telecommunications protocol. See Chorafas [Chorafas, 1984] and Livingston [Livingston, 1988] for details.

CHAPTER 2

DEFINITION, APPLICATION AND THEORETICAL BASIS

Prototyping, in the sense of a method of systems development, is an approach to systems development rather than a carefully delineated methodology. In this chapter, we will set the stage for our analysis of the developer's perspective of prototyping. To do so, we will proceed as follows. First, we will attempt a crude definition of prototyping. Second, we will trace the intuitive appeal of prototyping by seeing its use in other fields and in systems development specifically. Third, we will more carefully delineate the different facets of prototyping. Fourth, we will present a brief overview on variations on the basic concept of prototyping. Finally, we will set the intellectual stage for our study by placing the study in the larger context of MIS research and by examining the theoretical underpinnings of prototyping.

2.1 Definition

We define the prototyping method of systems development as follows: prototyping is a conscious attempt to deliver a running version of a system (or the major parts thereof) to a user very early in the development process. It is recognized that the system delivered is a rough draft, and can

CHAPTER 2

DEFINITION, APPLICATION AND THEORETICAL BASIS

Prototyping, in the sense of a method of systems development, is an approach to systems development rather than a carefully delineated methodology. In this chapter, we will set the stage for our analysis of the developer's perspective of prototyping. To do so, we will proceed as follows. First, we will attempt a crude definition of prototyping. Second, we will trace the intuitive appeal of prototyping by seeing its use in other fields and in systems development specifically. Third, we will more carefully delineate the different facets of prototyping. Fourth, we will present a brief overview on variations on the basic concept of prototyping. Finally, we will set the intellectual stage for our study by placing the study in the larger context of MIS research and by examining the theoretical underpinnings of prototyping.

2.1 Definition

We define the prototyping method of systems development as follows: prototyping is a conscious attempt to deliver a running version of a system (or the major parts thereof) to a user very early in the development process. It is recognized that the system delivered is a rough draft, and can

contain errors. What is sought is the user's reaction to the draft. The user's reaction to the draft guides the next version of the system. The process iterates until the user is satisfied.[1]

In so defining prototyping, we wish to draw a contrast with more conventional methods of systems development. With conventional methods, much effort is expended in analyzing the needs of the user and designing the system before the system itself is created. The user does not see the system until fairly late in the development process. As noted earlier, Henderson and Ingraham claim that up to 70% of the systems development effort precedes the first view of the system by the user [Henderson and Ingraham, 1982].

It is necessary to place an important restriction on the definition above. Organizations have information requirements on at least two distinct levels [Davis, 1982]. First, the organization as a whole has a general, "macro" level of information requirement. It is the task of the MIS as a whole to address these requirements. Second, individuals within the organization have task-specific, "micro" information requirements. It is the task of application programs to address this level of information requirements. It is this second level of information requirements that prototyping seeks to address.

The definition above corresponds generally to definitions of prototyping found in the literature. Consider the definition employed in the article containing the first printed reference to the term "prototyping":

> an initial and usually highly simplified prototype version of the system is designed, implemented, tested and brought into operation. Based on the experience gained... a revised... prototype [is] designed and implemented. The cycle is repeated as often as necessary to achieve a satisfactory operational system... [Bally, et al., 1977, p. 23].

Consider also the definition used in the article considered seminal in the field of prototyping:

> a system that captures the essential features of a later system... intentionally incomplete, it is intended to be modified, expanded, supplemented, or supplanted (italics in original) [Naumann and Jenkins, 1982, p. 30].

Two key concepts can be found in these definitions. First, prototyping is a conscious approach to the development of information systems. The developer intentionally is delivering the system early in order to provide a forum for feedback from the user. Second, prototyping is an iterative process. The developer recognizes that the process will be repeated several, if not many, times before the user's needs are satisfied. In fact, the developer should recognize that additional needs will be uncovered as the process iterates; indeed, it is the difficulty of uncovering the entirety of user needs that motivates the use of prototyping in the first place.

2.2 Prototyping in other fields

Though the term "prototyping" may not be used, many fields of human endeavor use the concept of modeling: architects commonly build scale models of a proposed building; physicians try out new surgical techniques on animals; statisticians run pilot studies to see if a given statistical

instrument is appropriate; businesses test-market new goods and services; and computer hardware engineers hand-build new circuitry before developing a production run of a new product.

Business disciplines have not ignored the concept of prototyping either. Cohen and Van Horn, for example, proposed in the early 1970's a series of laboratory experiments to examine various organizational development design alternatives. After establishing the validity of the study, they concluded that these "simulations can be designed so that they are ... prototypes of the system to be developed ... experimentation with prototypes can help to produce better system designs and lead to the introduction of a more effective system into the real world" [Cohen and Van Horn, 1972, p. 9]. A number of other examples of this approach can be found in management [Hayes and Nolan, 1974], in marketing [Urban and Karach, 1971], and in related areas of the computing literature [Albano and Orsini, 1984; Riddle, et al., 1978; Stemple, et al., 1985].

2.3 Early prototyping concepts in MIS literature

Although the term "prototyping" did not appear in the information systems literature until 1977 [Bally, et al., 1977], antecedent concepts appeared earlier. Ackoff called for the "participation of managers in the design of the system that is to serve them [in order to] assure their ability to evaluate its performance by comparing its output with what was predicted" [Ackoff, 1967, p. B-156]. Keen and Gerson argued for a form of prototyping development from a

political perspective [Keen and Gerson, 1977]. Lucas analyzed a conflict model for use/developer interaction and extensively discussed techniques to reduce communication difficulties. These techniques, which resemble prototyping in several important respects, were an attempt to enhance the creative aspects of conflict while minimizing the destructive aspects [Lucas, 1971]. Schewe and Wiek attacked the problem from a marketing perspective and concluded that in-depth user-involvement was a necessary condition of "selling" a new system to a user [Schewe and Wiek, 1977].

Many authorities have noted that human factors are key to acceptance and use of MIS [Carper, 1977; Maish, 1979; Smith, 1977]. There has been a growing acceptance of MIS as a complex organizational dynamic. Davis and Taylor addressed this by proposing a simulation method to obtain a clearer understanding of user requirements, a method which presaged much of prototyping's interactive nature [Davis and Taylor, 1975]. Frank advocated a "trial and error approach" to design [Frank, 1979], while Scott advocated a development process as "simply one of listening to their client's problem, attempting a solution, testing this with the client, modifying it for another try, and continuing with such interactions until they finally 'breadboarded' their way to the final solution" [Scott, 1978, p. 60].

Many authorities have sought to enhance user input, particularly in the development stages of an information system. Tersine and Riggs, for example, argue for explicat-

ing models to users to enhance developer understanding and user-involvement. In essence, they were recognizing that prototypes of the desired system are useful vehicles for discussion [Tersine and Riggs, 1976].

The central study of user-involvement is Lucas' book, Why information systems fail [Lucas, 1975]. Building on earlier work [Lucas, 1971], Lucas posits sixteen propositions regarding factors leading to the success or failure of information system. Most relevant to this dissertation is proposition 4: "User involvement in the design and operation of information systems results in favorable user attitudes and perceptions of information systems and the information services staff" [Lucas, 1975, p. 22]. Several of the studies described in the book confirm a strong positive correlation between involvement and appreciation of the system's potential, as had been hypothesized by Swanson [Swanson, 1974]. As noted by Cerveny and Clark, Lucas' points have become an ongoing point of discussion in the information community for many years, generating a number of research hypotheses [Cerveny and Clark, 1981].

There is thus clear motivation for pursuing prototyping as a means of systems development, and at the macro level the concept of "prototyping" is fairly well defined. It is at the micro level that no clear definition of prototyping exists, and that is perhaps all to the good given the field's immaturity. Nonetheless, in the following paragraphs we will attempt to give the term a clearer focus, concentrating on the definitions found in the literature.

Distinctions will be drawn along several dimensions: "rapid" versus more conventional prototyping; prototyping as modeling versus evolutionary development; prototyping as a requirements specification tool versus a development tool; and others.

Although written for the practitioner press, Johnson's article "A Prototypical Success Story" [Johnson, 1983] proposed a taxonomy of prototyping techniques that has been applied to more scholarly research, for example, by Cerveny, Garrity and Sanders [Cerveny, et al., 1986]. The Association for Systems Management also has adopted this taxonomy as an integrating theme for a series of seminars on prototyping [Association for Systems Management and Guimaraes, 1986]. Alternative taxonomies exist [Cerveny, et al., 1987; Davis, 1982, pp. 20 ff.; Guimaraes, 1987; Mahmood, 1987].

Johnson proposed a four-level taxonomy:

Level 1: Mock-ups: a manual form of prototyping with particular focus on input/output;

Level 2: Simulation: an automated form of mock-up, with input/output simulated but with no functionality included;

Level 3: Working model: an actual system with limited functionality; this level of prototyping implies that prototyping will be stopped at some point and the prototype discarded;

Level 4: Research and development: an evolutionary approach in which a complete system is developed and

evolves into an actual system; it is made distinct from a working model (level 3) in that the prototype itself becomes the production system.

Level 4 has other implications which we will pursue shortly.

Mock-ups (level 1) have been used by the information systems profession for some time; indeed, until recently, mock-ups constituted the bulk of systems analysis and design tools [Cougar, 1973]. Screen design sheets and report layout forms are examples of this class of tool. Simulation (level 2) is similar, and only slightly more advanced. "Slide shows" of successive screens in an interactive system are an example. A typical tool of this nature is Dan Bricklin's Demo Program [Bricklin, 1985]. Both classes have a common theme: the need to involve the user with the reality of the system-to-be. While both are abstractions, they are not far removed from reality. In this characteristic, they are distinct from many other systems analysis and design tools, for example, flowcharts. Neither of these classes constitute what is normally called "prototyping". However, the techniques they employ can be used beneficially as tools within a larger prototyping perspective.

For purposes of this dissertation, "prototyping" as a development technique begins with Johnson's level 3, that is, working models. As noted earlier, this concept has some heritage in the information systems literature [Tersine and Riggs, 1976].

Only recently have tools been available to quickly implement models [Sarvari, 1983], yet ironically, proto-

typing in conventional systems development environments may have been occurring all along. Fred Brooks, in his provocative book The Mythical Man-month notes:

> When a new system concept ... is used, one has to build a system to throw away, for even the best planning is not so omniscient as to get it right the first time. Hence, plan to throw one away; you will anyhow. (Italics in original.) [Brooks, 1975, p. 116].

Brooks was writing within the context of conventional systems development, and was speaking generally of large technical systems, specifically the development of OS/360. Yet, in his iconoclastic style, he was describing a practice that was, and still is, all too common in information systems development as well. Brooks' response to the situation described above was a series of management innovations, some of which have entered the mainstream of information systems development. Apparently, he felt that these management techniques would address the problems described. In view of his insights, it is interesting that he never carried the thought forward: to intentionally "build a system to throw away".

Brooks was addressing requirements analysis in the passage quoted above, and early work in prototyping focused on that phase. Johnson notes that levels 1 through 3, mock-ups through working models, are simply different tools to use within the larger context of the requirements definition stage of the traditional systems development life cycle [Johnson, 1983]. Gomaa and Scott, for example, focus on prototyping solely as a design tool, arguing that the

resulting system is so inefficient so as to preclude subsequent use as a production system [Gomaa and Scott, 1981]. Good et al. advocate a form of prototyping to adapt system interfaces to user needs, but present no concept of carrying the resulting code directly to the production system [Good, et al., 1984]. Connell and Brice clearly state that prototyping should be used in the even more restricted role of a requirements definition tool [Connell and Brice, 1983]. Many modern systems analysis and design texts continue this concept by restricting prototyping to a phase within systems design [Senn, 1984; Whitten, et al., 1986]. Weisman advocates prototyping using Artificial Intelligence techniques, but implies rewriting the design in COBOL [Weisman, 1987]. Boar's book Application Prototyping, one of only two text-length treatments of this subject, also treats prototyping as a subset of the design phase [Boar, 1984].

Major research on separate prototyping environments has been reported by Kruchten et al.. They describe SETL, a set-theoretic very high level language with a rich set of abstract primitives [Kruchten, et al., 1984]. They argue that such a system, divorced from any actual production development, can result in more powerful prototyping. SETL thus is designed solely for modeling activities.

Wasserman has contributed the "User Software Engineering Methodology", known as "USE", which involves the user in many stages of systems development but with particular attention to the user interface. Prototyping is advocated as one stage in a life cycle not dissimilar to the tradi-

tional systems development life cycle model [Wasserman, 1981; Wasserman, 1984]. A simulation sub-system ("RAPID/USE") has also been developed to facilitate this prototyping stage [Wasserman and Shewmake, 1985].

Others, without arguing for prototyping per se, advocate approaches that imply transfer to the prototyped system to the production system. McLean developed the concept of "throwaway code", which advocated building libraries of APL code to assemble quickly a system piecemeal [McLean, 1977]. Developers of modern programming environments have extended this idea. Goodman, for example, describes IMSADF as a facility for storing and reusing modules of code, with the resulting speedier development allowing a focus on application logic. Although not made clear, one presumes that the resulting assemblage in then compiled as a system [Goodman, 1980]. At the other end of the range of computing, Michielsen advocates a similar development plan for microcomputing, using the C language [Michielsen, 1986].

Burns and Kirkham describe a highly detailed and complex prototyping procedure that depends heavily on the concepts of modularity, locality of reference and portability found in the Ada language [Burns and Kirkham, 1986]. Their concept of prototyping is also that of modeling: a single-source input and a single-object output system is constructed as a single Ada program. The focus is on data flow within that program. Once developed, the system is then constructed.

Text-length treatments of systems analysis also treat prototyping as an evolutionary technique. Lantz, in one of only two books on this subject, advocates evolving the prototype into the final system, and notes modifications to the entire design process in order to accomplish this [Lantz, 1987]. James Martin's Application development without programmers treats a larger domain but does briefly address prototyping. Martin places evolutionary development on an equal footing with prototyping as a design tool [Martin, 1982].

Carey and Mason provide a particularly helpful overview of the prototyping literature, including analysis of the literature then existing along this and other dimensions [Carey and Mason, 1983].

Despite the activity described in the preceding paragraphs, most research on programming productivity has not dealt with the concept of prototyping. Reusable code and other innovative programming environment techniques that support prototyping concepts have not been dominant in professional thinking. For example, Hanson and Rosinski's recent article on programmer productivity does not mention the reuse of existing libraries and the implication that holds for developing systems quickly [Hanson and Rosinski, 1985]. In this study, programmers rated various tools in the production environment on their ability to make their jobs more efficient. Tools that would relate to reusable code (a private library, a program cross-referencer, etc.) ranked in the bottom half of a twenty-item list, well below

screen editors and printing utilities. This tells us that programmers have not yet taken to heart the prescriptions of those with a broader view of systems development.

2.4 Applications of prototyping

We now turn our attention to refining the definition of "prototyping" along several dimensions.

2.4.1 "Rapid" vs. "conventional" prototyping

A distinction can be made between prototyping that takes place in real time (generally called "rapid prototyping") and prototyping that takes place over several days and weeks. The literature, as a rule, does not make this distinction and it is left to the reader to determine what type of prototyping is discussed in a particular piece.

Most of the prototyping literature discusses prototyping that takes place over a period of days, weeks or months. Since there is no known term for this type of prototyping, the present author chooses to term it "conventional prototyping". While conventional prototyping is rapid with respect to normal systems development time schedules, this type of prototyping is not considered "rapid prototyping", which takes place in real time during a user-developer interaction. Conventional prototyping is well represented in the literature, particularly as case studies. Typical is Earl's "Prototyping Systems for Accounting Information and Control" [Earl, 1978; Earl, 1982]. Earl describes accounting system prototypes as being "designed and written quickly ... crude, rough and ready" intended to "test alternatives

designs through live operation" [Earl, 1978, p. 162]. Earl sees his prototypes as "working models", or as level 3 in Johnson's taxonomy.

It is clear from practical experience that accounting systems are complex and difficult to build, particularly if they involve managerial accounting. Help in refining the system to be built can be difficult since accounting decisions, particularly outside the world of financial accounting, are not at all clear cut. Yet, a crudely assembled system, such as Earl is advocating, would clearly be executionally inefficient, and the technology of the 1970's would have exacerbated this fact. Therefore, Earl sees prototyping as only a design technique. Gomaa and Scott's findings [Gomaa and Scott, 1981], previously discussed, confirm Earl's with respect to a complex system to coordinate the manufacture of integrated circuits.

Kraushaar and Shirland, in a comprehensive article, offer a state-transition model, noting that the "prototyping approach views the final operational system as the desired state that is achieved by passing through earlier, less desirable states" [Kraushaar and Shirland, 1985, p. 190]. Despite what this quotation implies, however, the final state is a separately developed system that uses the final stage prototype as a model. A more detailed, two-prototype methodology is endorsed, with the first prototype being solely exploratory and the second exploring output needs. It is clear that such a structured prototyping environment

cannot be considered "rapid prototyping". Other findings of this piece will be discussed later in this dissertation.

Mason and Carey propose an "architecture methodology" for prototyping. Although the bulk of their paper is devoted to the discussion of a specific prototyping tool, they propose a three-stage prototyping method similar in concept to Kraushaar and Shirland's. Again, a very structured prototyping method inhibits the responsiveness necessary for "rapid prototyping" [Mason and Carey, 1983]. Other implications of Mason and Carey's "architecture" approach will be discussed shortly.

On the other hand, support for rapid prototyping is documented in the literature. McNurlin cites several cases of rapid prototyping, including development within a single interaction session between developer and user [McNurlin, 1981]. Blum has published several case studies of prototyping [Blum, 1986A; Blum, 1986B]. Working within a medical environment, most of these were developed using TEDIUM, a system written in MUMPS and generating MUMPS target systems. This environment gives rise to the expectation that real time rapid prototyping is taking place. Blum views rapid prototyping as a design tool, however. It is a happenstance of his developmental environment that the prototype evolves directly into the production system.

2.4.2 Modeling vs. evolving

Closely related to the discussion above is the distinction between prototyping as a modeling technique contrasted with prototyping as a development technique. This distinc-

tion corresponds with levels 3 and 4 of the Johnson taxonomy [Johnson, 1983]. Blum puts the distinction more graphically [Blum, 1982]. He entitles the modeling approach "system architecture" and means by that term something close to Mason and Carey's meaning [Mason and Carey, 1983]. He entitles the evolutionary approach "system sculpture". In making this distinction, he notes that a single, comprehensive life cycle model is not applicable to MIS development; rather, a number of different life cycle models are needed, and that the sculpture model is useful within stated domains.

Necco et al., reporting on the opinions of senior MIS executives, argue that SDLC-based approaches will continue to dominate systems development. Prototyping "will be used increasingly to facilitate the definition of the users' requirements" [Necco, et al., 1987, p. 473]. These authors report a conservative "current wisdom" among senior MIS executives that prototyping is a modeling tool within the larger scope of the systems development life cycle.

Zelkowitz describes the development of an application programming system done as a rapid prototype [Zelkowitz, 1980]. He conducted a modeling exercise using SNOBOL4 for several iterations, followed by an implementation in Pascal. It is clear that the purpose of the rewrite was executional efficiency: had a more efficient language been used for prototyping, the final step may not have been necessary. Lirov and Daunov describe a similar approach, but even more

focused on simulating the final system, as opposed to development of a working, but slow model [Lirov and Daunov, 1985]. In both cases, the prototype was intended only to show the capabilities of the proposed system: at no point was it intended to evolve into the actual production system. Both these examples, therefore, exist on level 3 on Johnson's taxonomy [Johnson, 1983].

Zave, on the other hand, argues for the evolutionary approach to software, which she titles the "operational approach". In this approach, "during the specification stage, computer specialists formulate a system to solve the problem and specify this system in terms of implementation-independent structures that generate the behavior of the specified system" [Zave, 1984, p. 106]. After an appropriate series of transformations, "specifications structures can be mapped straightforwardly and efficiently onto a particular configuration of implementation resources" [Zave, 1984, p. 109]. Her work is data-structure-oriented, and exists as a compromise between Johnson's levels 3 and 4 [Johnson, 1983]. The data structures model the proposed system, and are not the production system, yet the mapping onto the production system is viewed to be incidental. Thus, we can see in Zave's work movement toward the evolutionary approach: her approach generates a series of behavioral expectations which evolve rather naturally into a production system.

Mason and Carey's "architecture methodology" paper [Mason and Carey, 1983] is primarily devoted to a discussion

of a specific prototyping tool, ACT/1. They propose a three-stage prototyping method. Although they state that "the architecture based method considered [the final] prototype to be ... 'version 0' of the system", in practice, "using [version 0] to execute the production version of the [interactive information system] requires increased (possibly substantial) computer systems resources" (emphasis in original) [Mason and Carey, 1983, pp. 349, 352]. Thus, while they propose a method based on Johnson's level 4 [Johnson, 1983], the limitation of existing technology forces them to employ the tiered structure implied by Johnson's level 3.

The practitioner press contains a series of evolutionary prototyping case studies and reports. Bottom et al. describe an evolutionary approach based on microcomputer database management systems [Bottom, et al., 1985]. Weisman applies the concepts of Artificial Intelligence to prototyping [Weisman, 1987]. McNurlin, as previously noted, cites several examples of evolutionary prototyping [McNurlin, 1981]. Boehm's empirical work [Boehm, et al., 1984], discussed later in this dissertation, also exists on Johnson's level 4 [Johnson, 1983].

2.4.3 Design vs. development tool

Most authorities view prototyping as a design technique [Henderson and Ingraham, 1982], with possible evolution into a production system. Some recent academic research has followed this line of reasoning [Sauter and Schofer, 1988].

A few, however, do not agree. Connell and Brice, analyzing a prototyping failure, reach the conclusion that "rapid prototyping is a requirements analysis technique, not a system development technique" [Connell and Brice, 1983, p. 523]. In fact, they argue, it is confusion on this point that lead to the failure cited in the first place.

Giddings [Giddings, 1984] concurs, noting that the purpose of the prototype is to establish the problem domain. The understanding gained is to be frozen at some point and taken to development. He thus views the prototype as a design tool. Meredeth makes similar points in the practitioner press [Meredeth, 1985].

On the other hand, many authorities view prototyping as advantageous as a development tool as well as a design tool. Contrary to Connell and Brice, Guimaraes [Guimaraes, 1983] points out that ongoing maintenance is a major, if not the dominating factor in overall system expense [see also Swanson, 1988]. Noting that in practice relatively little maintenance is corrective and most of it is perfective in nature, it makes sense to design the system well. It makes more sense to have an efficient method of changing the system not only during development but post-implementation as well. Developing strong user input via prototyping is a major step toward solving the ongoing maintenance problem, he argues.

Guimaraes echoes Lehman [Lehman, 1980], who notes, for certain classes of programs, maintenance expenses are so great that they overwhelm consideration of executional effi-

ciency; in these cases, it makes sense to leave the prototype as the final product, since ongoing changes (and the user interaction necessary to facilitate those changes) can be effected much more inexpensively in that environment.

2.5 Alternatives to prototyping

Within the macro definition of prototyping exists many kinds of evolutionary development techniques. Since at present we are addressing a definition at the micro level, we should make distinctions between prototyping as presented thus far and other, closely related developments.

A term closely associated with prototyping, and generally considered synonymous with prototyping, is "iterative development". Blum [Blum, 1986B] used the term in the sense of rapid prototyping. A few others make the distinction between iterative development as a general model as opposed to planned development.

The term "heuristic development" is not synonymous with prototyping, however. Conceived by Berrisford and Wetherbe [Berrisford and Wetherbe, 1979] in an effort to avoid "analysis paralysis" [Wetherbe, 1984], heuristic development differs from prototyping in one major respect: it is an output-first approach. The approach is data oriented, with a focus on the development of output reports and screens. The development of an input system is deferred to the end of the development [Berrisford and Wetherbe, 1979, especially page 16]. Prototyping, on the other hand, makes no such

distinction; in fact, one commonly develops an input system first.

Kraushaar and Shirland, previously discussed, offer a detailed comparison of the two development methods [Kraushaar and Shirland, 1985]. They argue for the development of the input system first for practical reasons: users respond better when confronted with real, as opposed to contrived data. Some system must be developed to populate the database with which the system will exist, as least with a selection of real records. Further, they argue, developing the input system forces the user to confront the issue of data to be collected by the system: this can be helpful in determining gaps or missing assumptions on the part of users or developers.

2.6 Theoretical basis of prototyping

We close this chapter by noting that prototyping, whether as a tool in the design stage of the systems development life cycle, as in Johnson's level 3, or as an alternative development model, as in Johnson's level 4 [Johnson, 1983], has firm theoretical grounding. Lehman provides such a conceptual foundation in a 1984 paper [Lehman, 1984], which builds on his 1980 paper cited earlier [Lehman, 1980]. He describes prototyping as "reification": the slow emergence of the actual needs and desires of the user from the morass of technical, social, political and emotional factors that plague the system development effort. This point is further explicated by DeSanctis and Courtney [DeSanctis and Courtney, 1983], who note the importance of cognitive and

emotional information shared by the user and the developer, as well as the factual and technical information.

Davis reaches similar conclusions, noting that within individual application software, two classes of needs exist: social and technical [Davis, 1982]. In making this statement, he must have been influenced by socio-technical systems development concepts [Edström, 1977; Fok, et al., 1987; Hedberg and Mumford, 1975]. Giddings pursues the same line of inquiry, noting the knowledge contained in application software is domain dependent: the developer must learn about the problem area as well as the software itself [Giddings, 1984]. It follows that the problem domains includes social and political dimensions as well as task-related knowledge.

Keen expands upon this concept, noting that the decision making process is relative, emotive and only partially cognitive [Keen, 1981]. When focusing on individual application systems related to tactical aspects of business operations, he advocates a development method very similar to prototyping. His motivation in doing so is to achieve success by overcoming the counterimplementation strategies of workers. The steps recommended ("seek out resistance and treat it as a signal to be responded to"; "rely on face-to-face contact"; "become an insider and work hard to build personal credibility") become a prescription for the prototyping method when applied to information systems development.

2.6.1 Applicable theoretical models

In turn, Keen's work in grounded in more general theoretical work, as is Alavi and Henderson's [Alavi and Henderson, 1981]. Three major theoretical models have been proposed for the examination of prototyping: the Lewin-Schein change model, the problem-solving model proposed by Mitroff et al., and the consulting model of Kolb and Frohman. The first two will be examined briefly, while the last will provide the basis for in-depth explication of the prototyping process.

The Lewin-Schein change model [Lewin, 1947] establishes a three-stage process of change. First, in the "unfreezing" stage, a "felt-need" for change must be established. Commentators note the need for the mutual commitment for this change on the part of both user and implementor of change (in the context of this dissertation, the systems developer) [Alavi and Henderson, 1981; Keen, 1981]. In essence, a contract for change must be established. This contract must be freely given, and based on mutual credibility. In the second stage, "change", the process in question is altered (in the context of this dissertation, a system is implemented). In the third and last stage, "refreezing", the change is institutionalized. Here the commitment developed in the first two stages is required to affirm the change, permitting it to be made concrete.

Mitroff's model [Mitroff, et al., 1974] envisions four stages. In the first stage, an initial analysis is performed and the problem is "conceptualized". In the second

stage, a "modeling" process takes place, with a crucial substage entitled "reality feedback". In the third stage, a solution is generated and results compared to those expected from the model chosen ("narrow feedback"). Although not detailed, one presumes the second and third stages are iterated if the results of the third stage are suboptimal. In the fourth and final stage, the resulting system is implemented.

The Kolb learning model [Kolb, 1979], which is a development and enhancement of the Lewin-Schein change model cited above, proposed the learning cycle illustrated in figure 2.1 (after Alavi and Henderson, 1981, p. 1321). The specific extension to the Kolb model to be considered here is the the Kolb-Frohman consulting model [Kolb and Frohman, 1970].

In the Kolb-Frohman consulting model, the creation of an attitude amenable to change ("unfreezing" in Lewin-Schein terminology) is divided into two substages: a "scouting" stage and a "entry" stage. The actual work of creating a system is seen as three stages: "diagnosis", "planning" and "action". These three stages correspond to the "change" phase of the Lewin-Schein model. These stages are iterated as often as necessary: at some iteration, "diagnosis" will indicate that no further action is necessary, the the process will then move onto the final two stages. Here, a final "evaluation" takes place, followed by a "termination" of the development process. The last two stages correspond

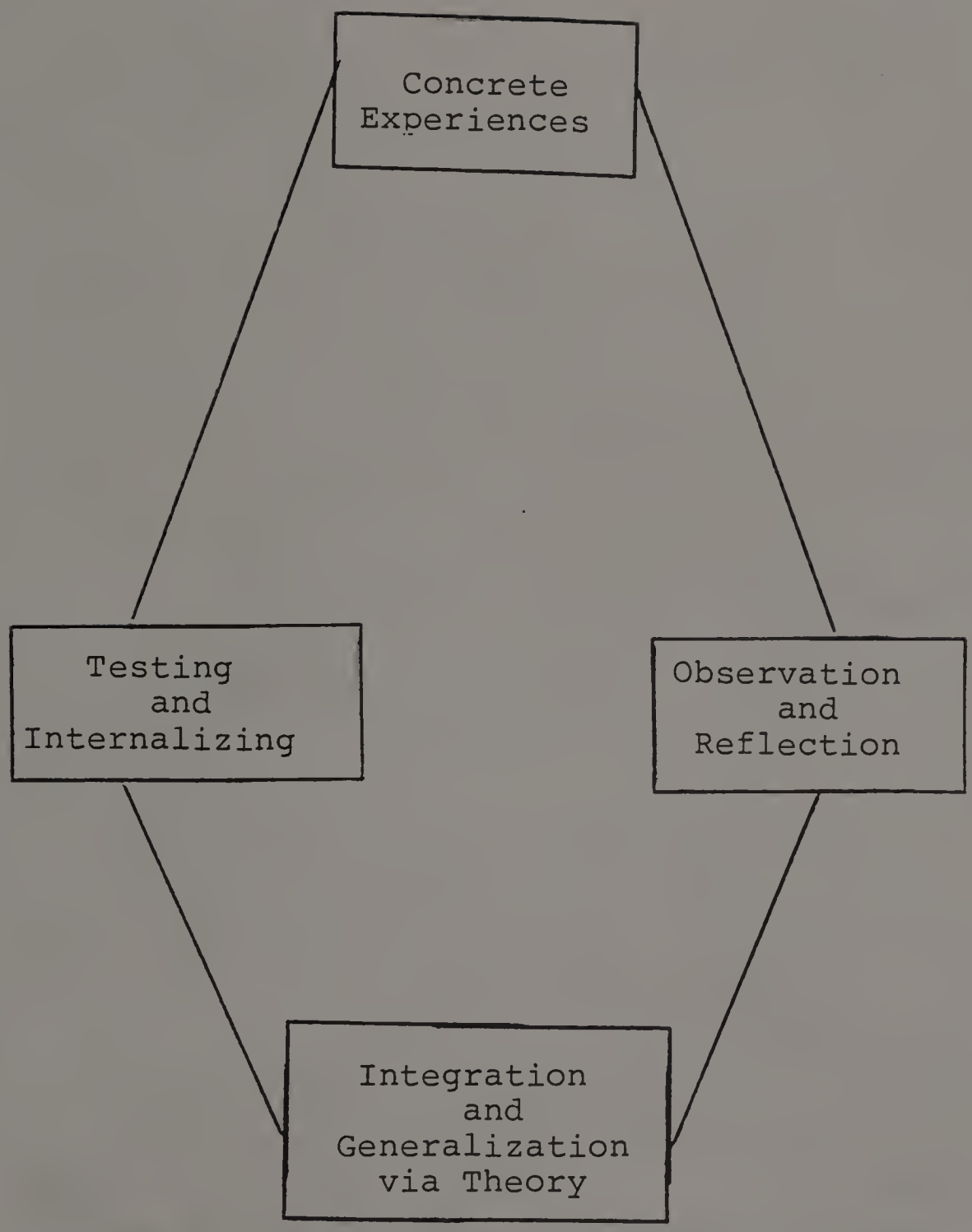


Figure 2.1

Learning Cycle Model

(after Alavi and Henderson, 1981, p. 1321)

to the "refreezing" stage of the Lewin-Schein model. Figure 2.2 (after Keen, 1981, p. 26) illustrates this.

2.6.2 Prototyping in MIS research

The study of prototyping as a particular form of user-involvement in systems development exists within the larger field of information systems research. Recall Mason and Mitroff's oft-quoted definition of an information system, as follows:

> at least one PERSON of a certain PSYCHOLOGICAL TYPE who faces a PROBLEM within some ORGANIZATIONAL CONTEXT for which he needs EVIDENCE to arrive at a solution (i.e., to select some course of action) and that the evidence is made available to him through some MODE OF PRESENTATION (italics and upper case in original) [Mason and Mitroff, 1973, p. 475].

The study of prototyping can also be placed in other research paradigms. Ives et al. have categorized MIS research in five major classes [Ives, et al., 1980]:

I: research on a single variable or group of variables;

II: research on relationships between process and environmental variables;

III: research between process variables and elements in the information system itself;

IV: research between environmental variables and elements in the information system; and finally

V: interactions among the variable groups.

We will address the place this research holds in Ives et al.'s paradigm momentarily.

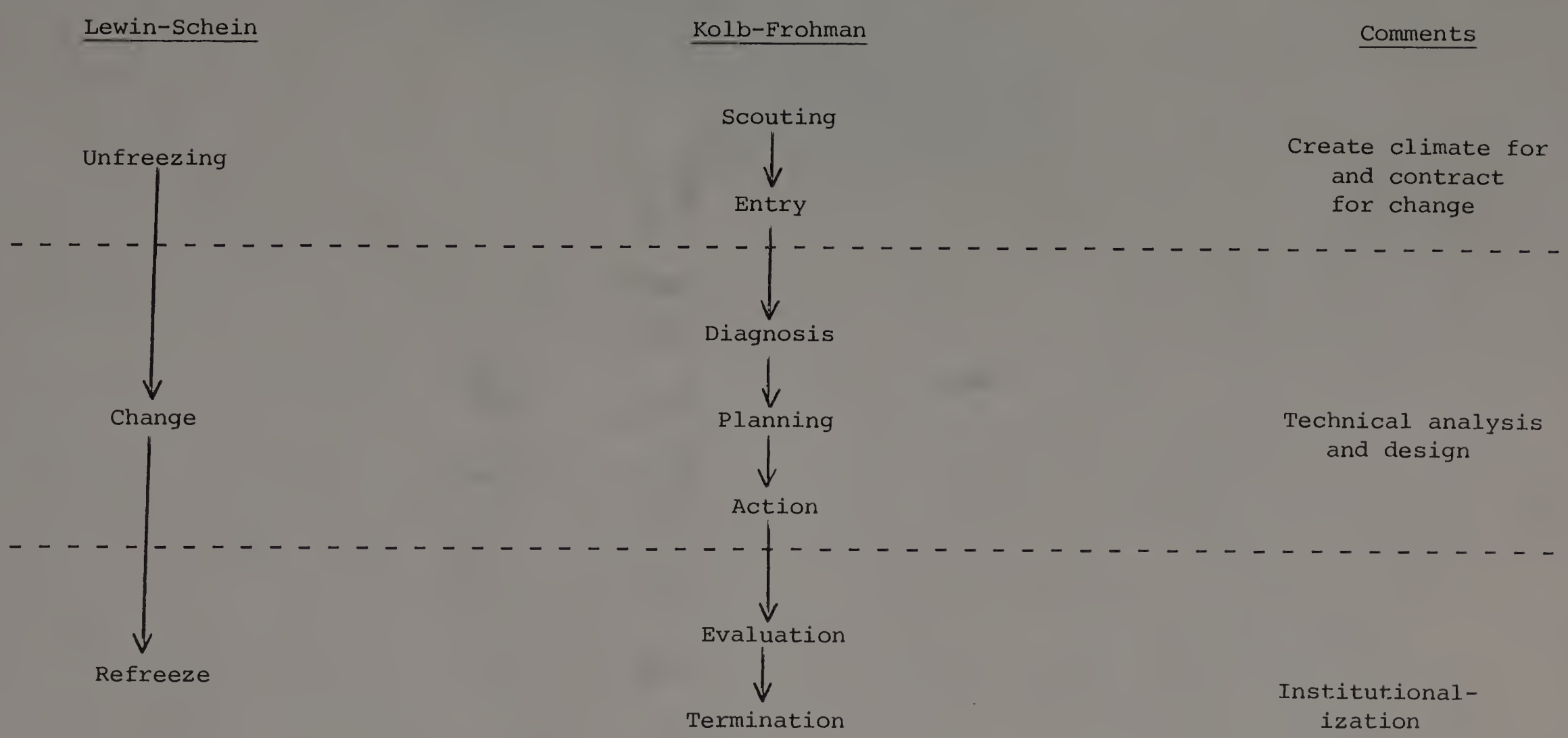


Figure 2.2

Comparison of Lewin-Schein and Kolb-Frohman models

(after Keen, 1981, p. 26)

2.6.3 Prototyping in scientific research

It will be enlightening to place prototyping within the larger body of scientific research. Kuhn [Kuhn, 1962] gives us a useful model, that of the paradigm shift. He argues that science can be seen as a series of theories, each in turn perceived to be an adequate model for explaining some phenomenon under discussion. As evidence gathers, however, doubt is cast on the current model, until such as time as the current model can no longer adequately explain the phenomenon. This state of affairs causes a disequilibrium, which in turn causes a new model to be posited which explain the new information. The resulting model then holds sway until additional information causes a new state of disequilibrium. This paradigm shift, he argues, is the guiding model of scientific inquiry.

Prototyping can be addressed using Kuhn's concept on two levels. First, in a macro sense, prototyping can be seen as a new paradigm posited to explain the well-known failings of the SDLC paradigm [Brooks, 1975; Swanson, 1988]. It is interesting to note the full title of the seminal article on prototyping: "Prototyping: the New Paradigm for Systems Development" [Naumann and Jenkins, 1982].

On the micro level, each iteration of the prototyping process is itself a new paradigm created to address the inadequacies of the previous iteration. The process continues until the present version eliminates, or at least dilutes, the factors causing the disequilibrium.

2.7 Conclusion

This dissertation will address the interaction of developers, users and the developmental process of information systems. Thus, in Mason and Mitroff's paradigm, we focus on persons and modes of presentation. First, we focus on persons because the development of systems is essentially one of human interaction rather than technical expertise. It is generally known how to develop large, complex systems (how well the job is done is open to question, however). What is clear from experience is this: it is not known how to handle well the human factors of complex systems. Second, within the context of systems development, it is clear that the mode of presentation used in the conventional systems development methods has not performed its task well.

In the paradigm of Ives *et al.*, the research reported here is of category III, that is, the relationship between process characteristics (type of applications, method of development) and particular variables of the information systems (speed of development, cost of development). We will also have occasion to address certain developer characteristics (satisfaction with system, perception of cost and speed of delivery). Since we are involving developer characteristics and thus a third body of variables, this research approaches type V in the Ives *et al.* paradigm.

Notes

1. The definition in this paragraph was used as the basis for the interview-based research described later in this dissertation.

CHAPTER 3

PROTOTYPING:
EMPIRICAL INVESTIGATIONS

Prototyping has not been subjected to extensive empirical analysis. What investigation has taken place has focused primarily on the user's perspective, and thus is tangential to the research focus of this dissertation, which is the developer's perspective. In this chapter, we will examine the empirical research of Alavi and Henderson, Boehm et al., Alavi, Mahmood and Necco et al.. These five studies collectively constitute the bulk of empirical study on prototyping. For each study, we will overview briefly the research and subsequently examine the findings of the research with regard to developer perspectives.

3.1 Alavi and Henderson, 1981

Alavi and Henderson [Alavi and Henderson, 1981] conducted a laboratory experiment in which students learned and subsequently implemented a reporting system based on an established quadratic cost function model. Several implementation strategies, including evolutionary development, were employed and these, together with the decision style of the user, constituted the independent variables. Decision style criteria were in turn based on the psychological types provided by Jung [Jung, 1971], although not all of Jung's

dimensions are included in the research reported. The dependent variables were usage of the resulting decision support system ("DSS") and user satisfaction with the DSS. As can be seen clearly from the variables under consideration, this study focused very closely on the user and his/-her perceptions.

Perhaps the most significant finding of Alavi and Henderson's study was this: "the findings support the hypothesis than an evolutionary implementation strategy is more effective than a traditional strategy" [Alavi and Henderson, 1981, p. 1320]. This finding supports the efficacy of prototyping. Only in the conclusion of the paper do Alavi and Henderson address the implication of their findings regarding developers. Based on the finding regard decision-making style of users (findings not discussed here), they advocate that designers consider the nature of the learning activity inherent in an evolutionary development. Based on the findings regarding the superiority of the evolutionary approach, the authors note that the designer must consider the sequential process of the prototyping approach. They conclude by noting that "the results suggest the impact of the learning process in DSS design and the role of the DSS designer as a learning process facilitator will be important areas for future research" [Alavi and Henderson, 1981, p. 1321].

3.2 Boehm, et al., 1984

Boehm *et al.*'s research was also a laboratory-based experiment [Boehm, *et al.*, 1984; see also Boehm, 1984], but was more akin to MIS development than was Alavi and

Henderson's. Boehm *et al.* were also interested in the prototyping process *per se* and did not limit themselves to the user's perceptions, as did Alavi and Henderson. He had seven teams of graduate student's develop a user-interface to COCOMO, a model for software estimation [Boehm, 1981]. Teams consisted of either two or three students. Four of the teams adopted conventional analysis and design techniques (which Boehm *et al.* term the "specifying approach") while three used the prototyping technique. The teams engaged in conventional development were mandated to produce requirements and design documentation, an operational program, a user's manual and maintenance documentation. The prototyping teams were required to produce a workable prototype by the mid-point of the semester. All programs were coded in Pascal.

The resulting programs were evaluated along four major dimensions: functionality, robustness, ease-of-use and ease-of-learning. A number of interesting finding came out of the research. While the programs of the specifying teams were rated higher along the functionality and robustness dimensions, the prototyped programs were superior along ease-of-use, ease-of-learning and maintainability dimensions. There was an insignificant difference with respect to actual productivity of the teams.

It can be seen that the focus of this experiment was the development process itself. Nonetheless, Boehm made a number of statements related to the value of prototyping to

developers. For example, he noted that "the process of prototyping gave software developers a more realistic feel for the amount of effort required to add features to a project... the lack of a definitive specification meant that prototypers were less locked into a set of promised to deliver capabilities than were the specifiers" [Boehm, et al., 1984, pp. 298-299]. Boehm et al. emphasized that prototyping is not more productive, in the sense of delivering more lines of final code per worker-day. "However, if 'productivity' is measured in equivalent user satisfaction per man-hour, prototyping did tend to be superior" [Boehm, et al., 1984, p. 299]. If we grant the reasonable assumption that developers desire to please users, this is powerful motivation indeed for prototyping.

The fact that prototyped systems were judged superior along the maintainability dimension is also a powerful motivator. This is even more so if the developer him/herself, or his/her organization, will be responsible for maintaining the system in the future.

3.3 Alavi, 1984

Alavi reported additional research in 1984 [Alavi, 1984]. The research was in two phases: a field-interview survey of twelve information systems developed using prototyping and a laboratory experiment. The two phases will be discussed separately here.

Alavi interviewed twenty-two systems professionals: twelve project managers and ten systems analysts. Thus, the findings of this portion of Alavi's research are applicable

to this dissertation's focus on the developer's perception of prototyping. A number of findings are relevant. The most important is: prototyping is recognized as an alternative method of systems development by these working professionals. They have very favorable attitudes toward the method: MIS professionals find that it provides a common base line from which to generate expectations for the system and engender user enthusiasm. The MIS professionals did find, however, that maintaining a high level of user enthusiasm can be difficult.

Alavi also found that MIS professionals perceived that prototyping engenders better developer-user relationships and, perhaps most crucial, "prototyping gets it right" [Alavi, 1984, p. 558].

Alavi also found that prototyping is most applicable to small-scale systems. This finding confirms the opinion of the Boehm et al. study [Boehm, et al., 1984]. Contrary to these findings, however, are reports in the literature demonstrating the efficacy of prototyping-like approaches to large-scale system development [Edelman, 1981].

The second phase of Alavi's research was a laboratory-based experiment. Sixty-three M.B.A. students participated in the experiment. The subjects were drawn from evening graduate students; ninety-two percent held full-time professional positions. In this experiment, the independent variable in the experiment is the type of development approach employed: prototyping versus a life-cycle-based conventional

approach. Three classes of dependent variables were examined. Only one of these classes, the perceptions and attitudes toward the design process by both users and developers, is relevant to the research interest of this dissertation.

The focus of this portion of the experiment was on user's perceptions. Users exposed to prototyping and users exposed to a conventional life-cycle-based development methodology were compared. Users exposed to prototyping were significantly more satisfied with the resulting system and with their participation with the development of the system. Users exposed to prototyping also perceived significantly less conflict with developers.

Developer's perceptions were reported for two factors. Developers using prototyping perceived marginally less control over the development process and significantly greater frequency of change in user requirements. Both of these findings are self-evident given the nature of prototyping. The present author feels the first finding is obvious: the very purpose of the prototyping process is to generate user response. This response inevitably takes the form of requests by users for changes. This is perceived by designers to be changes in user specifications.

3.4 Mahmood, 1987

A recent report of empirical research on prototyping is Mahmood's [Mahmood, 1987]. This research was based on sixty one pairs of returned mail questionnaires: in all cases, one response came from users in an organization and one from a

systems professional. The research interest of this dissertation mandates that we address only the results based on the systems professionals. The majority of system professionals represented were directly involved in development (62%), while only 23% were in management positions or did not report their positions; 15% were characterized as system development managers.

The analysis of designer responses focused on whether a SDLC-based development method or prototyping was preferred along twenty-four dimensions. Significant difference in terms of the method preferred were found in eleven of the twenty-four dimensions, leading to the overall conclusion that "respondents did not clearly favor either SDLC or the prototyping approach" [Mahmood, 1987, p. 298]. Of the eleven dimensions on which significant differences were found, prototyping was significantly preferred by designers in terms of:

1) satisfaction with user participation;
2) [a reduction of] user/designer conflict;
3) extent of user use of the system;
4) satisfaction with the development approach;
5) flexibility of the approach;
6) validation of user requirements;
7) implementation and user acceptance; and
8) acquisition of expensive software.

A conventional SDLC-based approach was preferred along the following dimensions:

1) ease of project management and control;

2) project completed on schedule; and

3) ease of systems planning.

Mahmood also examined the combined results of both designer and user groups. Here, the results generally corresponded with the results reported in the preceding paragraphs: the conventional SDLC-based approach was significantly favored in terms of project management and prototyping was significantly favored in terms of perceived user contributions.

Since so little research has been conducted on developer's perceptions of prototyping, Mahmood's study can be considered groundbreaking. He did, however, confirm Alavi's findings [Alavi, 1984] that designers perceived prototyping to be a less structured environment, leading to more changes in user specifications. The present author submits that this finding is self-evident given the nature of prototyping.

Mahmood's research was descriptive in nature, although he did attempt to build a framework for reviewing and selection a development methodology for a given project. He does not address the issue of motivation, however: why do systems designer prefer prototyping, as opposed to conventional approaches? Previous research also fails to address this issue.

3.5 Necco, et al., 1987

The research reported by Necco et al. was primarily descriptive in nature, for they attempted to assess the

current state of systems development practices [Necco, et al., 1987]. They reported that a large majority of MIS executives whose organizations reported using prototyping discovered changes sooner; that systems were developed in less time; and that the user was more satisfied with the resulting system. In all three cases, a majority found the results to be of "major importance". Interestingly, the forty-three executives reporting were unanimous on the last point: that users are more satisfied with systems developed using prototyping. A majority also reported that prototyped systems were developed less expensively, but the number assessing this to be of major importance was split about evenly with those who found this result to be of minor importance [Necco, et al., 1987, p. 471].

While the results reported by Necco et al. are interesting, it must be carefully noted that they reflect the opinions of senior MIS executives and not systems developers themselves. While these findings will support some contentions reported in this dissertation, as a body they are tangential to the research interest here. Therefore, the findings of Necco et al. will not be further considered in this dissertation.

3.6 Conclusion

Chapter 2 reported on the general run of the MIS literature as it relates to prototyping, while this chapter concentrated on empirical research. Taken together, it is clear that most of the research work on this subject has

been descriptive and theory-building [Bally, 1977; Naumann and Jenkins, 1982]. This should not be a surprise, given the newness of prototyping and the general immaturity of the MIS discipline.

Within the scope of software prototyping, few of the published pieces can be identified as research, and fewer still can be identified as empirical research. The five research reports constitute virtually the entire body of empirical research on software prototyping. After examining this body of research, it is clear that what empirical research has taken place has had as its focus the prototyping process itself [Boehm, et al., 1984], the user's perspective of the prototyping process [Alavi and Henderson, 1981; Alavi, 1984] or managerial issues [Mahmood, 1987; Necco, et al., 1987]. Within the last group, much attention has been given to decision models to discern whether or not prototyping is an appropriate development model for a given project.

Lacking has been the developer's perception of the prototyping process. Especially lacking has been research into why a developer chooses the prototype when confronted with a system development project. The research reported in the next two chapters will address this issue.

CHAPTER 4

RESEARCH HYPOTHESES AND RESEARCH DESIGN

This chapter will address the research hypotheses to be examined in this dissertation. The primary research purpose of this dissertation is to explore the developer's perceptions of the prototyping process, within the scope of research outlined in section 1.5. We are particularly interested in exploring the motivating factors for developers to use the prototyping method of systems development. The chapter closes with a discussion of the research design and techniques employed.

4.1 Research hypotheses

As noted in the previous chapter, prototyping is a relatively unexplored field. The field is particularly unexplored with respect to analyzing the developer's perception of the prototyping process, especially the developer's motivations for adopting prototyping. In this section, we will state four main research hypotheses, each representing a dimension of the developer's perception of prototyping. All hypotheses will be stated in alternate form, that is, the hypothesis will be asserted if there is significant evidence to support it; otherwise, it will be considered false. We will briefly state the conclusion reached with

each hypothesis, but will reserve detailed discussion of the conclusions until chapter 5.

All research must begin with clear definitions of relevant terms, and the research reported here is no exception. Chapter 2 reported extensively on the use of the term "prototyping" as it appears in the literature. It was noted that there is broad consensus on a macro definition of prototyping, but that at the application level there is a considerable domain of activity that can be called "prototyping". The present author desires to explore if the same consensus of definition exists in the research population. This interest gives rise to the first hypothesis (note that this and subsequent hypotheses are stated in alternate form):

> Hypothesis 1: There is no broad consensus of opinion in the research population on the definition of the term "prototyping".

Note that this hypothesis tests the congruence of the respondents' understanding of prototyping. Therefore, like a goodness-of-fit test, it is not desired to assert this alternate hypothesis. In the research reported in chapter 5, we do not assert this hypothesis, thus concluding that there is a broad consensus on the definition of prototyping.

Is prototyping faster than other means of development? There is some evidence in the literature that prototyped systems require fewer programmer-hours of development effort than conventionally developed systems [see, for example, Boehm, et al., 1984; see especially Figure 1, p. 293]. The

present author feels that this research focus is somewhat misplaced: from a client's perspective, the programmer-hours of development are less important than the actual time it takes to receive one's system. If developers comprehend this point, they will be interested in the calendar time a system requires from conception to delivery. This reasoning gives rise to the second hypothesis, stated in alternate form:

> Hypothesis 2: Developers perceive that prototyped systems can be delivered in less calendar time than systems developed in the conventional manner.

In the research reported in chapter 5, we will fail to assert this hypothesis, concluding that prototyped systems are not delivered in significantly less time than systems developed conventionally.

Are prototyped systems less expensive than systems developed in the conventional manner? Systems development is a competitive business, and the research sample chosen (independent entrepreneurs for the most part) is particularly sensitive to this competition. This sensitivity gives rise to the third hypothesis, also stated in alternate form:

> Hypothesis 3: Developers perceive that prototyped systems can be delivered at less cost than systems developed in the conventional manner.

In the research reported in chapter 5, we will not assert this hypothesis, concluding that prototyped systems are not

provided at significantly less cost than systems developed in the conventional manner.

If, as our research will indicate, systems developers perceive that prototyped systems are not delivered in significantly less time or at significantly less cost than systems than systems developed in the conventional manner, why do systems developers choose to prototype? In part due to the competitive environment of independent systems developers, and in part due to other businss factors to be discussed later in this dissertation, we posit that developers are interested in delivering the best quality systems that time and resources permit. This gives rise to the following, fourth hypothesis, stated in alternate form:

Hypothesis 4: Developers perceive that prototyped systems are of higher quality than systems developed in the conventional manner,

In chapter 5, we will discuss our research finding that asserts this hypothesis. In doing so, we will conclude that systems quality is a major motivator for systems developers to adopt the prototyping approach. This in turn will give rise to good deal of discussion and interpretation.

4.2 Research design

The research reported here is based on a series of twenty-nine indepth, probing interviews with software developers who consciously employ the prototyping method of systems development.

4.2.1 Respondent qualifications

All respondents were qualified on several factors:

1) all respondents know about and consciously use the prototyping method of systems developement;

2) all respondents prototype primarily MIS application software, although occasionally undertaking other types of systems;

3) all respondents operate independently, whether as independent entrepreneurs or as independently operating staff members within organizations large or small;

4) all repondents concentrate their development efforts in the microcomputer and small minicomputer environments;

5) all respondents utilize features of database management systems and/or software libraries as prototyping tools; and,

6) all respondents participated voluntarily in the study described here, with no compensation whatsoever.

4.2.2 Sample development

During the summer of 1987, very considerable efforts were undertaken to identify potential subjects. Using lists of consultants developed by user groups [Boston Computer Society, 1987], directories of authorized consultants published by database management system software vendors [Data Language Corporation, 1986; Microrim, 1986], and the author's contacts in industry, a list of forty-seven potential respondents was developed.

All potential respondents were contacted by telephone. In that conversation, the general objectives of the research were described. The intent of the initial conversation was to determine whether the individual was sufficiently aware of prototyping and whether s/he used prototyping as a development technique. Twelve potential respondents were eliminated from the sample on the basis of this initial contact, leaving thirty-five potential respondents. The remaining potential repondents received a letter confirming the telephone conversation and indicating the researcher's intention to contact them in the fall of 1987 to establish an appointment for the research interview.

All of the twelve potential respondents eliminated on the basis of the initial conversation received courteous letters thanking them for taking the telephone call and noting that they would not be included in the survey.

Of the thirty-five potential respondents remaining, two were subsequently eliminated from the survey because the focus of their systems development work changed during the approximately five months between the initial telephone contact and attempts to interview them for this research. Conversations with these individuals clearly indicated that the change in focus was a function of the consulting work they had retained in the interim (a major contract designing a telecommunications network in one case, supported graduate research in Artificial Intelligence in the other). In the judgement of the present author, there was no mortality bias

associated with the departure of these two subjects from the research sample. Their departure was acknowledged with a courteous letter. Thus, thirty-three potential subjects were available for the research sample.

It was not possible to establish appointments with three of the thirty-three remaining potential respondents, although it was verified that all three are actively involved in systems development. In one case, the subject did not return a number of telephone messages; in another case, it was not possible to establish an appointment convenient to the subject and the researcher; in the third and last case, an appointment was established, but a crisis at a client's site outside the immediate geographic area caused the respondent to be absent when the researcher arrived for the scheduled interview. In the view of the author, no mortality bias is associated with these three respondents not being included in the research sample. Thus, thirty respondents were available to the author.

All thirty respondents work in the New England area (one is located in Albany, New York). A very large majority are located in the greater Boston and greater Hartford areas. Following initial qualification during the summer of 1987, as described above, all thirty respondents were contacted by telephone during the fall of 1987. Though scheduling was difficult at times, all thirty agreed to be interviewed for the research. Once an interview appointment was established, a confirming letter was sent. During the conversation that established the appointment, the respondent's

commitment to be interviewed was reaffirmed. Other matters, such as road directions to the meeting place, were also discussed.

In ten of the thirty cases, the researcher was delayed in establishing an appointment. To sustain interest in the research and to assure the respondents that they had not been forgotten, letters were mailed in November, 1987, assuring respondents not contacted at that point of the researcher's continuing interest in them as research subjects.

The researcher then proceeded to interview the thirty respondents. Twenty-nine respondents proved to be articulate, thoughtful and informative. One respondent was so inarticulate and rambling that the researcher had great difficultly understanding what the respondent was trying to communicate. That interview was dropped from consideration. Thus, the research sample consists of twenty-nine indepth interviews. To preserve anonymity, the twenty-nine subjects are identified as Respondent 1 through Respondent 29. Their transcribed responses to twenty-one key questions are attached as Exhibit C through Exhibit AE of the Appendix.

4.2.3 Evaluation of the interview

The twenty-nine usable interviews took place between October 27, 1987 and January 6, 1988. The interview that was dropped from consideration took place on January 4, 1988. The interview instrument is attached as Exhibit A of the Appendix. Following completion of the interviews, the

results were compiled and analyzed. The instrumentation form is attached as Exhibit B of the Appendix. The results of this analysis make up the research results reported in the next chapter.

4.2.4 Sample size determination

The sample size was determined using a standard formula. First, it was determined that the majority of statistical tests would be qualtitative, primarily proportion difference tests[1]. Second, the level of significance was established at 0.10. Third, it was further determined that the tests need not be sensitive to proportion differences below 0.15. Assuming a population proportion of 0.5, these conditions set the minimum sample size of 31. Subsequent work established the minimum sample proportion to be approximately 0.6. Using this figure as a point estimate of the population proportion, the sample size could drop to 29 and still satisfy the conditions described above. Thus, when the sampling approach described above caused the sample size to drop to 29, no attempt was made to increase the sample size to compensate.

Notes

1. Specifically, it was envisioned that the normal approximation of the binomial distribution would be the basis for testing. The requisite conditions that $(np > 5)$ and $(n(1-p) > 5)$ were fulfilled in all cases.

CHAPTER 5

RESEARCH FINDINGS

This chapter details the results of the research project described in the latter part of the preceding chapter. We begin by examining the research sample of twenty-nine independent software developers from a demographic perspective. We then proceed to give a detailed accounting of the research findings, including tests of the four hypotheses described in the first portion of the preceding chapter. Discussion is included as appropriate. A general discussion concludes the chapter. For a faster review of the research findings, the reader is referred to the conclusion contained in section 5.2.5 and the interpretation contained in section 5.3.3.

5.1 Demographic analysis

The research interview began with a number of questions concerning the respondent's background (see Exhibit A in the Appendix). These questions had two purposes. First, by beginning with questions of this nature, it was anticipated that the respondent would be put at ease. The interviewer's perception is that this occurred in all twenty-nine cases. Second, these questions would support the present demographic analysis.

The respondents as a group can be characterized generally as young, male and prosperous. Their average age is estimated to be 37.2 years, with respondents ranging from an estimated 22.5 to 57.5 years. Seven of the twenty-nine are in their twenties, ten in their thirties, ten in their forties and two in their fifties. Twenty-eight of the twenty-nine are male. No questions were asked regarding the respondents' financial situation, but the interviewer did meet over half of the respondents in their home offices. Judging anecdotally based on this and other obvious factors, it can be safely concluded that the respondents are prosperous.

The respondents are also well educated. Twenty-eight of the twenty-nine are high school graduates, while the one high school dropout is obviously well self-educated (he is, in fact, a published author). All twenty-eight high school graduates attended college, and twenty-seven of them achieved the bachelor's degree (one holds an associate's degree). Fifteen of the twenty-seven bachelor's degrees holders went on to graduate school, and fourteen of them earned a graduate degree. Thirteen hold a master's degree while one (a former college professor) holds a doctorate.

The respondents are well experienced. They report a mean of 10.9 years of professional experience, and range from six months to thirty years. Given the relative youth of the members of the sample, the experience reported represents a significant amount of programming/analysis experience (a mean of 4.7 years), project management experience

(a mean of 2.7 years) and lesser amounts of other management experience (a mean of 1.4 years). The respondents also have substantial amounts of other technical experience (a mean of 1.9 years) and a smattering of training experience. Descriptive statistics of these factors can be seen in Table 5.1. Examination of the quartiles will reveal that there is a large amount of positive skewing on several factors.

The respondents' work experience can also be characterized generally as follows: the respondents had roughly equal experience in administrative versus non-administrative roles (a mean of 4.9 and 5.9 years respectively) and roughly equal experience in mainframe computer and microcomputer environments (a mean of 4.8 and 4.2 years respectively). A far smaller degree of experience in minicomputer environments was also reported (a mean of 1.9 years). Both the mainframe and minicomputer experience factors exhibit a large amount of positive skewing, as seen in Table 5.1. Microcomputer experience, on the other hand, exhibits neutral skewing. It is interesting to note that the mean of 4.2 years of microcomputer experience roughly equates to the phenomenon of business microcomputing. As a group, the respondents have been been involved in business microcomputing since the beginning of the phenomenon.

Dating the phenomenon of application software prototyping is considerably more difficult than dating the phenomenon of business microcomputing, but we can get some clues from two facts: the first published reference to the

Table 5.1

Demographic Analysis

Factor	Mean	Std Dev	Min.	Q_1	Median	Q_3	Max.
Estimated age	37.2	8.3	22.5	27.5	37.5	42.5	57.5
Year began prototyping	1976.1	7.9	1958	1968	1979	1982	1987
Years of experience	10.9	7.6	0.5	5	8	16	30
Years programming experience	4.7	4.2	0	1	3.5	7	16
Years of project management experience	2.7	4.1	0	0	0.5	4	18
Years of management experience	1.4	2.7	0	0	0	2	12
Years of technical experience	1.9	3.3	0	0	0	2	15
Years of mainframe experience	4.8	7.2	0	0	0	5	25
Years of microcomputer experience	4.2	2.1	0	3	4.5	5	8
Years of non-administrative experience	5.9	4.2	0	2.5	4.3	9	16
Years of administrative experience	4.9	5.2	0	1	2.3	8	24
Years aware of prototyping	8.1	6.0	1	3	6	10	23
Years using prototyping	5.4	4.9	0	2	4	6	20
Number of systems prototyped	22.3	35.6	0	5	8	15	150
Percent using prototyping, development role	60%	30%	0%	30%	66.5%	90%	100%
Percent using prototyping, non-development role	40%	30%	0%	0%	32.5%	70%	100%

term "prototyping" in 1977 [Bally, et al., 1977] and the publication of Naumann and Jenkins' seminal article in September, 1982 [Naumann and Jenkins, 1982]. In this context, it is very interesting to note that respondents claim to have been aware of prototyping as a mean of systems development for a mean of 8.1 years, and to have consciously used the technique for a mean of 5.4 years. The positive skewing seen in Table 5.1 can be explained. Several respondents noted prototyping-like experience in other fields, for example, in marketing. There is also some confusion regarding the definition of "prototyping", as will be seen shortly. This helps to explain why some respondents "prototyped" ten years before the term gained currency in the MIS field.

The experience of the sample can also be seen in the number of systems they reported to have developed, a mean of 22.3 systems. Of the twenty-eight respondents reporting a figure, one subject reported no systems developed at all while, at the other extreme, one subject reported 150 systems developed. Further, the respondents reported using prototyping in over half the systems they developed (a mean of 60%), and to have been involved with prototyping even with systems in which they held no development role (a mean of 40%).

The respondents for the most part are independent entrepreneurs (eighteen of twenty-nine, or 62%). Ten subjects were full-time MIS employees of organizations, though two of them maintain small consulting practices on the side.

These ten were evenly divided between large and small organizations. Discussion revealed that all ten function independently, and their day-to-day activities follow an entrepreneurial model. Only one has a full-time job not immediately related to information systems, and he develops systems as a part-time consulting practice. Figure 5.1 illustrates the job mix of the twenty-nine respondents.

5.2 Research findings

Following acquisition of the demographic information presented in the preceding section, the substantive portion of the interview began.

5.2.1 Commonness of definition

The first item of business was the definition of the term "prototyping". Given the relative uniformity of definitions in the term in the literature, at least in the macro sense (see the discussion in chapter 2 and the introductory definition in the interview instrument, Exhibit A of the Appendix), it was anticipated that there would be a strong commonness of definition in practice. This proved to be the case, but with some major qualifications.

Clearly, the majority of respondents agreed with the macro definition of the term as presented in chapter 2. Respondents were requested to give the interviewer their "working definition" of the term "prototyping". Not surprisingly, the respondents' definitions had a strong practical bent. Typical is the following comment:[1]

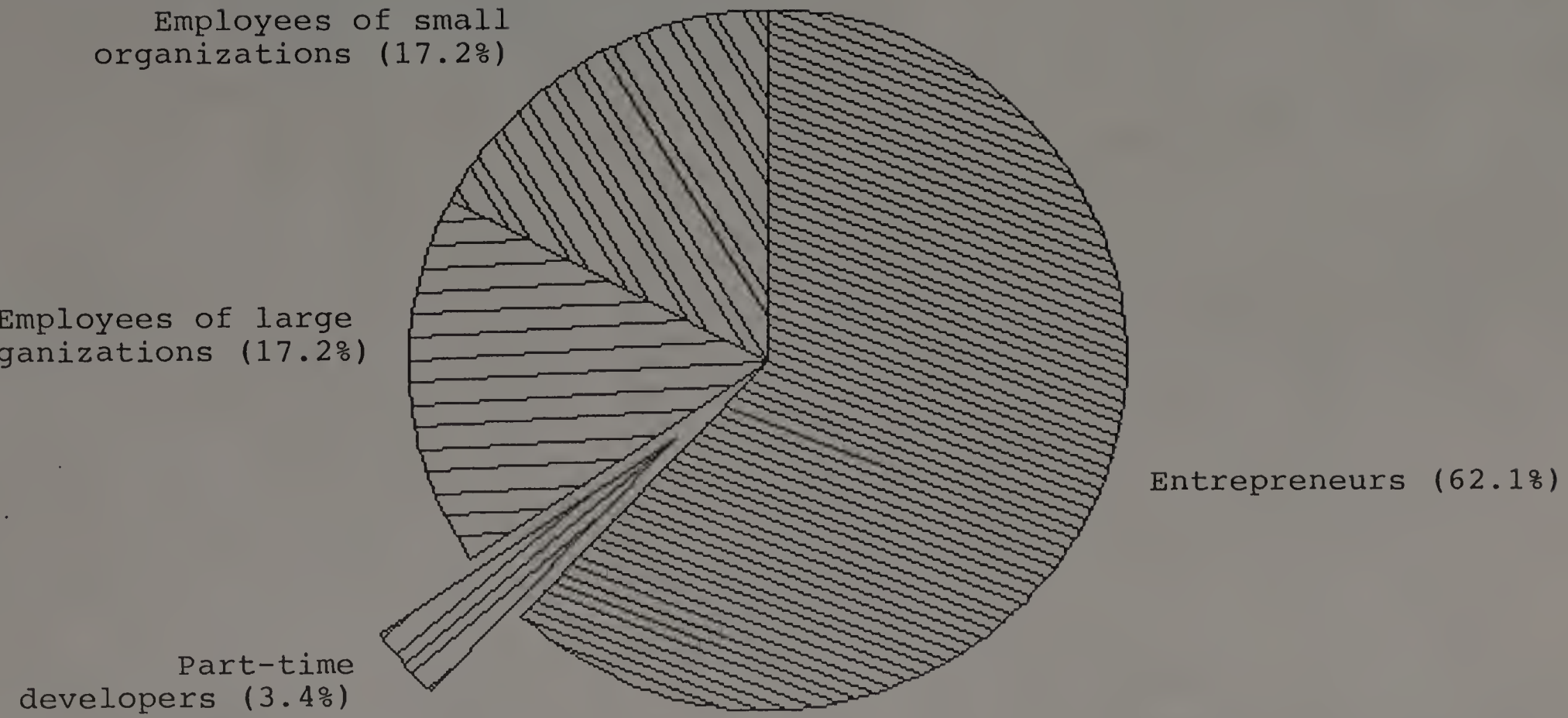


Figure 5.1

Respondents by type of employment

(n = 29)

> It's a mockup. With some assumptions in mind we put together what we perceive is a functional prototype system - what we perceive that to be. And the only way you can elicit comments from the user community is to give them something to comment on. So, I guess the functional definition of a prototype is just that: it's a working product of what was discussed...

Even more succinct is the following functional definition:[2]

> Prototyping is a way of getting a client to visualize the end product without having to develop the whole thing. ...It's a feedback mechanism for users.

Simultaneously, however, the term "prototyping" can mean somewhat different things to others. Several respondents perceived prototyping to be a marketing tool. Not surprisingly, these respondents focused their businesses on the development of commercial packages in contrast to custom-developed software. The following quotation epitomizes this perception:[3]

> My definition is perhaps easier to look at from a hardware standpoint, or some device, some widget, that you develop a prototype to try to sell it: try to sell the prototype, not necessarily what it looks like, in its final form, but here's the prototype of this particular gadget, and it doesn't work and now we need money to develop it. From the software side... the only time we would follow that strict definition would be if we were developing a product without a known customer... But, we perceive a need in the marketplace and have the time and the money to develop some kind of a product... We would put together the way the system would look overall, with perhaps most of the screens, for example, developed. A few of the output reports in place. One of the major functions of the system probably fairly close to being at least our version of being complete, with the rest sort of hanging out there, tempting on the menu... We would try and capture as many commitments as possible. Basically, if we were able to complete this to your specification, Mr. Customer, will you commit to buying it if we do that?

Another respondent argued similarly:[4]

> Prototypes for use are that stage after it [a custom-developed application program] has [been] developed for client A, when we try to cut away those things which

> are personality-specific to him [client A] and to add those things which we feel would be useful to others and then present it to clients B, C and D.

Still other respondents view prototyping strictly in the simulation sense, that is, level 3 in the Johnson taxonomy [Johnson, 1983]. For example:[5]

> ...its a matter of very quickly fabricating essentially a shell, the visual appearance of the system, which is able to demonstrate what the functionality of the system will be ultimately.

Similarly, developers are faced with the task of designing a system for use in environments other than the one in which the system was developed. Prototyping comes into play in this instance, in which a respondent proposed a two-part definition:[6]

> There are two types of prototyping. There's prototyping that's for the purpose of evolving a system. Then there's prototyping for the purpose of emulating an existing system on better equipment.

Another respondent held a definition of prototyping that was not shared by any other. This respondent developed software of a particularly technical nature, primarily for evaluating computer system performance. This different focus may have flavored his definition:[7]

> The kind of prototyping I do... is more of an internal process. I will go through the prototyping cycle for myself, and part of this is a reflection of the fact that in most of the projects I've dealt with, I've had a lot of leeway in designing the user interface... I'm kind of the end user there, although not really the end user, and I will do my own internal prototyping... that's my style of development.

Subsequent conversation indicated that this respondent also prototyped with clients in a way closer to the commonly accepted definition of the term.

Finally, one respondent seemed not to clearly understand the modern meaning of the term "prototyping". Responding to the introductory definition of prototyping (see chapter 2 and Exhibit A), he said:[8]

> Prototyping as you [the interviewer] have defined it was a requirement that we had to have when we had programming languages like COBOL and BASIC and even before those... We had to work with the users and... try to freeze the specification and program it.

When the researcher noted the disparity of this definition with the commonly-accepted meaning of the term, the respondent continued:

> Now, when we get into the fourth generation [language] world, we started operating more like a company operates... When that type of capability came along, we could go to users [and easily modify data structures, especially files]... So, we could do it earlier in the cycle in terms of writing programs... Therefore, you can go to the user earlier and say 'Let's make our mistakes together'...

The present author concludes that this respondent has a operating definition of the term prototyping that is fairly congruent with the accepted meaning of the term. The confusion arises from his labeling of past development practices as "prototyping", as if the term is synonymous with "development".

Indeed, it can be stated with certainty, based on this sample, that the research population had a generally accepted definition of the term "prototyping" that is congruent with the published definitions of the term. What is new in this research is the additional feature of the term as applied to marketing concepts.

Given this background, we can proceed to test Hypothesis 1. The "test" here will not be in the formal statistical sense, unlike subsequent tests in this chapter. Rather, we seek here to establish whether there exists sufficient consensus of opinion to conclusively state that there exists a common definition of prototyping. We begin by repeating the hypothesis, stated in alternate form:

> Hypothesis 1: There is no broad consensus of opinion in the research population on the definition of the term "prototyping".

Twenty-five of the twenty-nine respondents (86%) indicated agreement with the interviewer's introductory definition of "prototyping" (which is identical to the definition given in chapter 2 of this dissertation). Although respondents had their own refinements to the concept, most particularly in the marketing area, there exists broad consensus of opinion as to what is meant by the term "prototyping".

It is recognized that this consensus of opinion could be a function of social factors. It should be remembered that the definition provided by the interviewer was given in the introduction of the interview, and the respondents were asked for their "working definition" of prototyping early in the interview. We desired to seek an indicator of the importance of prototyping in the professional lives of the respondents. To that end, a series of questions followed regarding the short-term and long-term professional goals of the respondents, and the role prototyping would play in achieving those goals. To our surprise, a binomial response

occurred: respondents either saw a substantial role for prototyping in obtaining these goals, or saw no role for prototyping at all. A few responsdents indicated the role of prototyping in obtaining their goals was unknown. By far, the respondents indicated prototyping played a substantial role in obtaining their short-term professional goals (twenty-two of twenty-nine, or 75%) and long-term goals (eighteen of twenty-eight, or 64%). Figure 5.2 illustrates these findings. Given the demonstrated importance of prototyping in the professional lives of the respondents, as well as its currency in the systems development field, we conclude that respondents have undoubtedly considered the subject extensively. Therefore, they have good reason to have developed reasoned and detailed "working definitions" in their minds. The consensus of opinion achieved, which fails to assert Hypothesis 1, is indicative of a strong homogeneity of understanding of the term "prototyping".

5.2.2 Time dimension

A subject of continuing interest is the time benefit to be derived from using the prototyping method of systems development. Previous research has indicated that prototyped systems are achieved in somewhat fewer programmer hours of development time than equivalent systems developed in the conventional manner [Boehm, et al., 1984]. The time dimension explored in this dissertation is not programmer-hours of development but calendar time from conception of

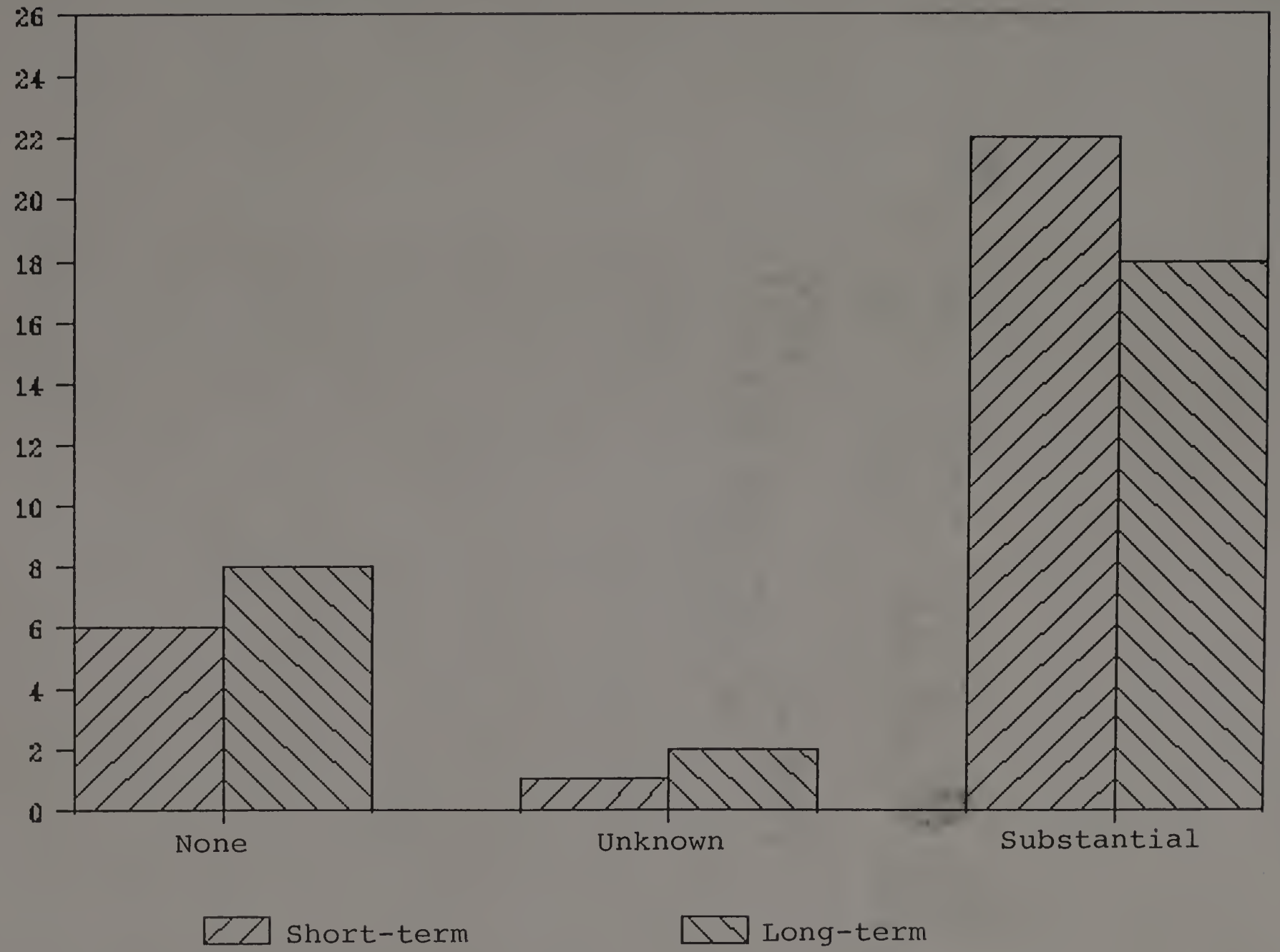


Figure 5.2
Perceived career impact of prototyping
(n = 29)

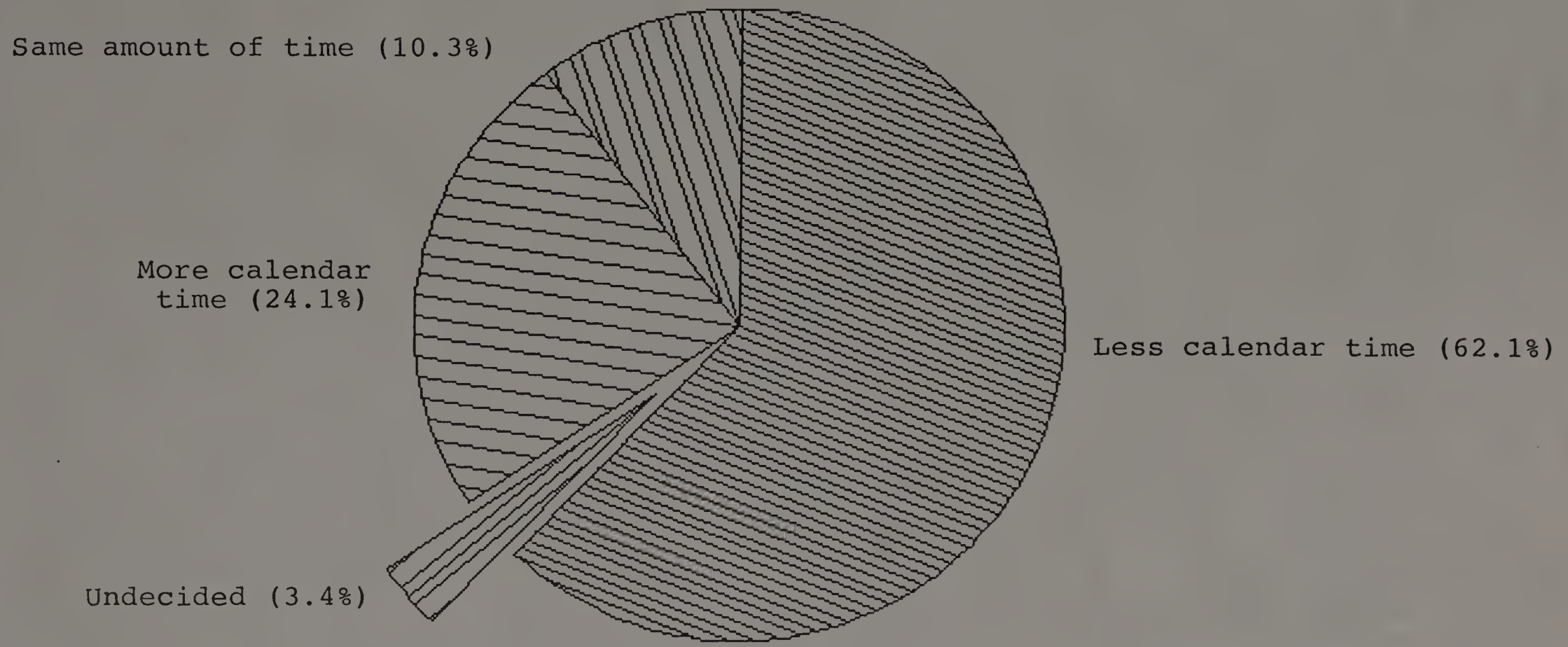
the system to its delivery. This is expressed in the following hypothesis, stated in alternate form:

> Hypothesis 2: Developers perceive that prototyped systems can be delivered in less calendar time than systems developed in the conventional manner.

The research shows that eighteen of twenty-nine respondents (62%) indicated their opinion that prototyped systems can be delivered in less calendar time than the same system developed in the conventional manner. A further seven (24%) indicated that prototyped systems took longer to deliver, while three (10%) felt there was no significant difference in delivery time. One respondent was undecided. Figure 5.3 illustrates these results.

Applying a standard statistical test of proportion difference to this data reveals muddled results. Note that the assumptions stated at the conclusion of chapter 4, namely, a 0.10 level of significance and an assumed population proportion of 0.60, are employed. Applying the usual monotonic transformation to the sample proportion of 0.621 ("standardizing"), we achieve a standard score of 0.23 ($p < 0.4090$), indicating an insignificant difference. Thus, we fail to assert Hypothesis 2, concluding that developers do not perceive that prototyped systems are delivered in significantly less calendar time than conventionally-developed systems.

The muddled results appear when we alter the assumptions and examine the sample size. If we assume an unknown population proportion, we must use a assumed population



98

Figure 5.3

Time dimension

(n = 29)

proportion of 0.5 in the standardization. Under these circumstances, the resulting standard score is 1.33 ($p < 0.0918$), which is significant at the 0.10 level of significance. Further complicating the picture, note that the sample size of twenty-nine is small, particularly for qualitative studies. Since small sample size mitigates against significance, we are left with the conjecture that the difference possibly would be significant in a larger study and with the parameter assumption altered.

To gain further understanding of this phenomenon, we asked respondents if they were currently involved in a specific prototyping development project. Twenty-one of twenty-eight[9] (75%) responded that they were. All twenty-one indicated that this was a typical project. The remaining seven were asked to keep a specific typical prototyping project in mind. A brief description indicated that all twenty-eight specific projects were well within the domain of MIS application software.

Respondents were then asked to picture in their minds this same project developed in the conventional manner. Twenty-five claimed they could do so. The respondents were then asked to describe how much longer or shorter the development time of this specific prototyping project was compared to the same project developed conventionally. The researcher then converted the response to a percentage ratio by treating the imagined conventionally-developed system as 100%. Seventeen of the twenty-five (68%) indicated that the prototyped system was delivered in less calendar time. This

proportion is insignificant at the 0.10 level of significance ($p < 0.1949$). Twelve of the twenty-five (48%) indicated the prototyped system was delivered in half the time. Four respondents (16%) indicated no significant difference in delivery time, while an additional four (16%) indicated that the prototyped systems took longer to deliver than conventionally developed systems. Figure 5.4 illustrates these findings.

Treating the imagined conventionally-developed system as 100%, the mean response was that prototyped systems are delivered in 72% of the calendar time that conventionally-developed systems require. We can gain additional insight if we momentarily take liberties with statistical testing procedures and treat this ratio as a quantitative variable. The 72% figure is significantly less than the 100% figure at the 0.10 level of significant ($p < 0.0036$, $n = 25$). The liberties taken in gaining this insight are considerable: the Central Limit Theorem calls for the population distribution to be symmetric and approximately normal in order to invoke the Theorem for this sample size. To test the "approximately normal" distribution of the data, a standard goodness-of-fit chi-square statistic for this data was calculated to be 24.68 ($p << 0.01$). Since the desire in a chi-square goodness-of-fit test is not to reject the hypothesis that the data follows the normal distribution, this result calls for considerable caution in inference. It is presented here solely for the purpose of gaining insight.

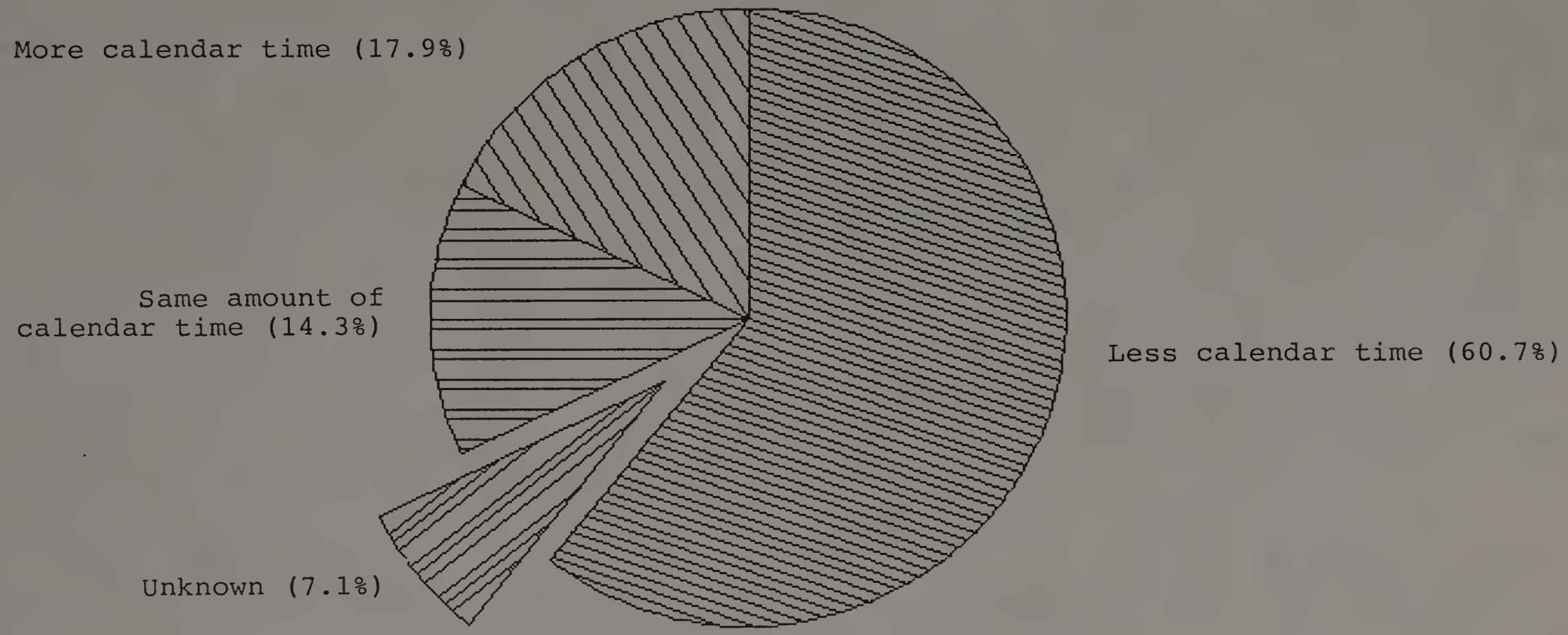


Figure 5.4

Specific project time perception

(n = 28)

Having failed to assert Hypothesis 2, our research indicates that systems developers do not perceive that prototyped systems are delivered in less calendar time than systems developed in the conventional manner. Our findings, however, are not clear-cut. Clearly, further research is required on this point.

A reasonable question is: do developers who view prototyping as a marketing tool feel differently on the time issue than do developers who primarily develop custom software. Two respondents (Respondent 11 and Respondent 28) prototype exclusively in a marketing mode. If we exclude them from consideration, we find that seventeen of twenty-seven (63%) of respondents who feel that prototyping delivers systems in less calendar time than conventionally-developed systems. This proportion is insignificant ($p < 0.3745$, $n = 27$). Interestingly, this proportion is not significantly different from the same proportion of the full sample ($p < 0.4721$).[10]

Additional understanding can be gained by cross-tabulating the respondents' perceptions on the time dimension with their type of employment.[11] Twelve of eighteen (67%) of the full-time entrepreneurs felt that prototyped systems could be delivered at less cost than conventionally-developed systems, while six of the remaining eleven respondents (who were employed by organizations) felt the same way (55%). Neither of these proportions is significant, nor is there a significant difference between the two proportions ($p < 0.2578$). (Note, however, that subdividing the sample

in this way reduces the statistical power of these tests.) We can conclude that entrepreneurs and non-entrepreneurs feel the same way on this issue. These results are presented in Table 5.2, together with results discussed in the next two sections of this chapter.

We conclude this section by sampling respondent opinions on the time dimension issue. Transcribed responses to the time-oriented question can be found in the interview transcriptions, included in this dissertation as Exhibit C through Exhibit AE, inclusive, in the Appendix. As noted above, respondent opinion can be divided into three areas: the majority opinion that prototyped systems can be delivered in less calendar time than conventionally-developed systems; the contrary opinion, namely, that conventionally-developed systems can be delivered in a shorter period of time; and those that feel there is no significant difference between the two development environments with respect to calendar time to delivery.

The majority opinion can be represented by the following quotation:[12]

> [Prototyping] makes the system-building process concrete. The analysis and discussion phase can waste a lot of time arguing abstractions.

Similarly,[13]

> If you mean successfully delivered, I would say yes... because clients don't know what they want, and they won't know what they want until they've had a chance to see it.

The distinction between "delivery" and "successful delivery" is one to which many respondents refer. The respondents

TABLE 5.2

Cross-tabulations of respondent employment with time, cost and quality dimensions

Time dimension:

	Employment status: Full-time Entrepreneur	Other type of employment	Total
Time response			
Less time	12	6	18
More time	2	5	7
Same time	3	0	3
Undecided	1	0	1
Total:	18	11	29
Proportion for less time:	.67	.55	

Cost dimension:

	Employment status: Full-time Entrepreneur	Other type of employment	Total
Cost response			
Less cost	13	6	19
More cost	3	4	7
Same cost	2	0	2
Depends	0	1	1
Total:	18	11	29
Proportion for less cost:	.72	.55	

Quality dimension:

	Employment status: Full-time Entrepreneur	Other type of employment	Total
Quality response			
Equal quality	1	2	3
Greater quality	7	4	11
Total:	8	6	14
Proportion for greater quality:	.88	.67	

seem to understand intuitively the fact that most maintenance is not of a corrective nature [Swanson, 1988]. As the person responsible for the overall success of the project, they realize that their task is not completed until the client expresses satisfaction with the system. The following quotation makes this point clear:[14]

> It depends on what you call 'delivered'. In a traditional life cycle, I can imagine two points in time. I can imagine delivery and sometime later when it's [the system] really working. And, I would imagine that the delivery time of a prototype system would fall somewhere in between, that there might be a little extra calendar time involved in actually getting the thing initially installed, into production. But, it would be more likely to be really workable at that point in time, and not require lots of hassles until you get it in place...

This point is discussed in some detail in section 5.3.3.

Other respondents perceive no major difference between the delivery time of prototyped and non-prototyped systems. For example:[15]

> ...if the customer wants something prototyped, and you give it to them prototyped, it takes X amount of programming to produce that. If they don't want it prototyped, then it's up to me to give them something, and generally it takes about the same amount of time, because programming is programming.

Still others were willing to sacrifice delivery efficiency (if, indeed, prototyping is less efficient with respect to conventionally developed systems on this dimension) in order to gain other benefits. As one respondent reported:[16]

> ...in my perception, what you're getting from prototyping is largely quality and accuracy and satisfaction with the ultimate product. I, generally speaking, don't think it's any more efficient than any other [development] method, from the point of view of calendar time...

Similarly,[17]

> There is a certain amount of inefficiency that comes from trying something, going back, the back-and-forth, I think, [that] may actually consume some calendar time. I don't think that's a bad thing. I think in the end you probably end up with a much stronger system that will last longer [and] be good for a longer time.

Apparently, those developers perceive prototyping to be not significantly more efficient with respect to delivery of the system in calendar time because either: one, they intuitively understand the real impact of perfective maintenance on their workload; and/or, two, they view the quality issue as paramount, so that they are willing to sacrifice time efficiency to achieve it.

5.2.3 Cost dimension

A subject related to the discussion of time dimension above is that of the cost dimension. As noted in Chapter 4, the respondents work in a competitive environment. A substantial proportion of the research sample is self-employed. Delivering systems at the lowest cost possible is understandably a strong motivator for the respondents. Previous research does not address the cost dimension, but the previously-reported findings that systems can be delivered in fewer programmer hours, together with the better maintainability ratings associated with prototyped systems [Boehm, et al., 1984], allows a reasonable inference that prototyped systems are less expensive to develop than are conventionally-developed systems.

We intend to pursue the developers' perception of the cost of systems. We will not examine these findings in the

same amount of detail as we did the time issue in the preceding section, largely because the techniques employed are repetitious. Our third hypothesis, stated in alternate form, is:

> Hypothesis 3: Developers perceive that prototyped systems can be delivered at less cost than systems developed in the conventional manner.

Nineteen of twenty-nine (66%) respondents felt that prototyped systems can be delivered at less cost than conventionally developed systems. Six (21%) that felt prototyped systems cost more while three (10%) felt there is no significant difference with respect to cost. Figure 5.5 illustrates these findings.

Standardizing this finding, and employing the same assumptions as in the previous section, the 66% proportion is insignificant at the 0.10 level of significance (its standardized score is 0.604, $p < 0.2617$). This mandates that we fail to assert Hypothesis 3, concluding that developers do not perceive prototyped systems to be delivered at significantly less cost than conventionally developed systems.

If we alter the parameter assumption, a muddled result appears, as occurred with the time dimension above. Under these circumstances, we state that the population parameter is unknown; we must then use an assumed population proportion of 0.5, and the sample proportion reported above becomes significant ($p < 0.0475$).

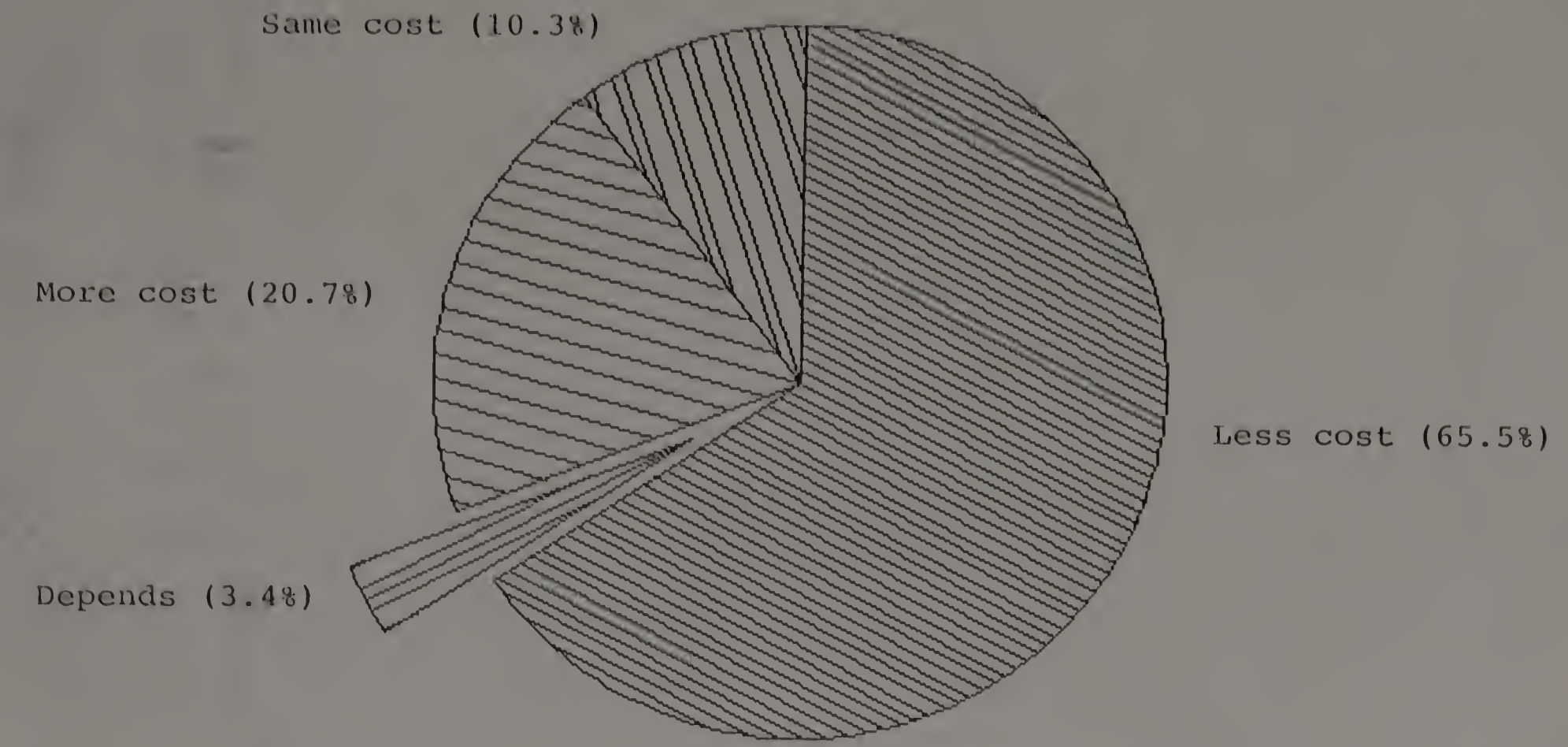


Figure 5.5

Cost dimension

(n = 29)

As was the case with our examination of the time dimension, additional insights can be gained by analyzing the developers' view of a single typical project. The project examined is the same one analyzed with respect to the time dimension in the previous section. A majority of developers (57%) felt that the specific prototyped system under consideration was delivered at less cost than the same system developed conventionally. However, as this is less than the assumed population proportion of 0.60, this proportion is obviously not significant. This finding is presented as Figure 5.6.

The researcher then queried the developers' perspectives on the ratio of delivery cost. Fourteen of the twenty-six usable responses (54%) indicated a ratio less than one, that is, that the prototyped system was delivered at less cost than the same system developed conventionally. This proportion is insignificant. Six (23%) perceived the two development methods to be equal on this dimension (that is, a ratio of 100%) while another six (23%) perceived the prototyped system to cost more.

We again can gain understanding if we let the imagined conventionally-developed system represent 100%, and quantify the developer's response as a ratio to that imagined system. If we take the same liberties with statistical testing procedures as we did in the preceding section (noting again the caution regarding the poor goodness-of-fit to the normal distribution found there), that is, treat the ratios expressed as a quantitative variable, then an interesting result

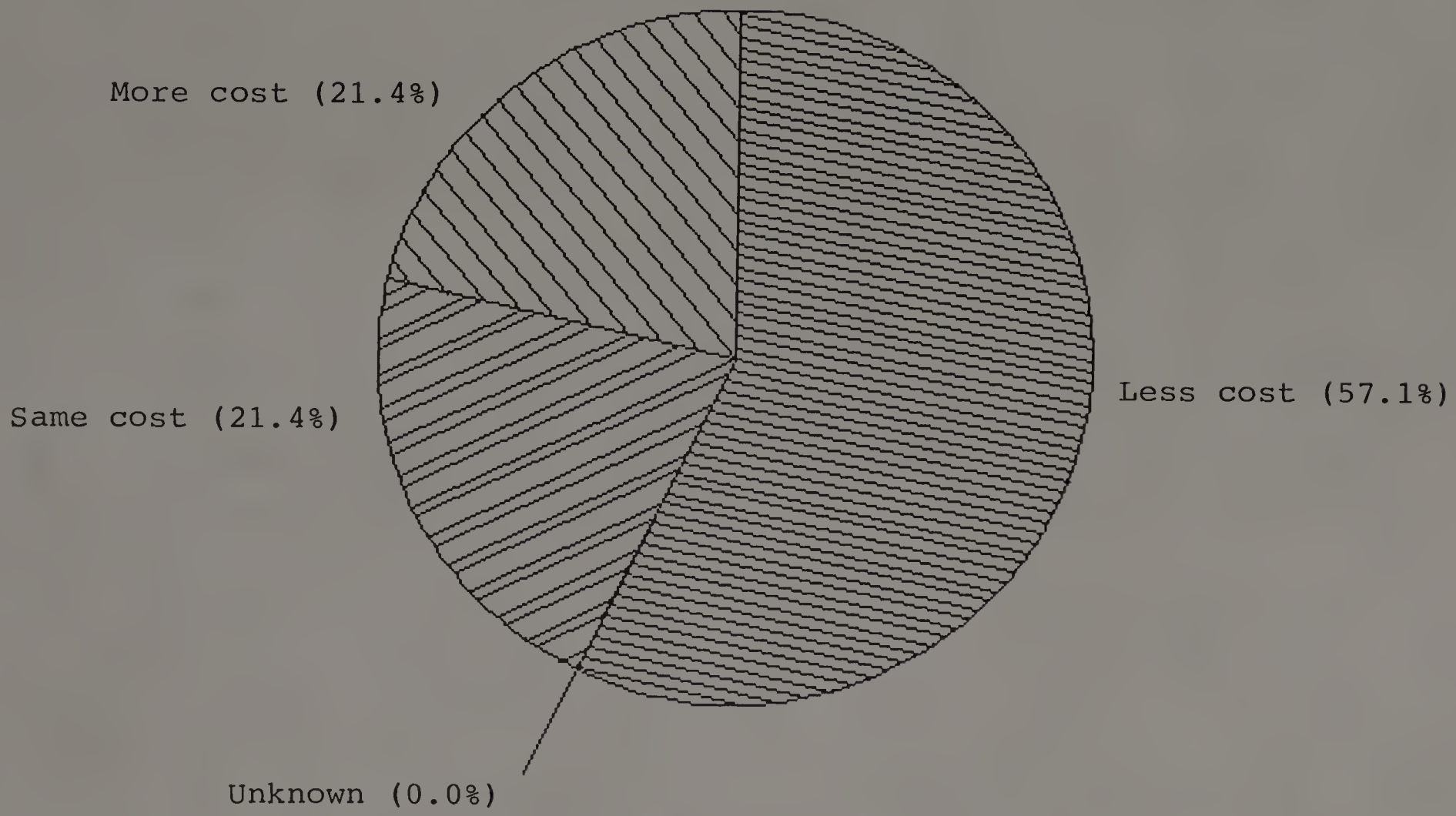


Figure 5.6

Specific project cost dimension

(n = 28)

occurs. The mean ratio of 85% is significantly less than the 100% ratio representing equality, at the 0.10 level of significance ($p < 0.0823$).

As was done with the time dimension, we will gain insight if we cross-tabulate the type of respondent's employment with his/her perception of the cost dimension. Thirteen of eighteen full-time entrepreneurs (72%) reported that they perceived that prototyped systems can be delivered at less cost than systems developed in the conventional manner, while six of the eleven other respondents concurred (55%). Neither of these proportions is significant, nor is there a significant difference between the two groups ($p < 0.1635$), but note again the caution generated by the low statistical power caused by small sample sizes. This leads us to conclude that the perceptions of entrepreneurs on this dimension are not significantly different from those employed by organizations. These results are presented in Table 5.2, previously presented, together with findings discussed in the next section of this chapter.

In addition, we cross-tabulated variables representing the time and cost dimensions. Not surprisingly, respondents who believe that prototyping delivers systems in significantly less calendar time also believe that prototyping produces systems at significantly less cost. Fifteen of the eighteen respondents who believed that prototyped systems are delivered in less time also believed that prototyped systems are delivered at less cost. Conversely, five of the seven respondents who believed that prototyping required

more calendar time also believed that prototyped systems require more cost. Table 5.3 contains details of these findings.

Further insights still can be gained by examining some of the respondents' reactions to this line of questioning. The responses to the cost-oriented question are found in Exhibit C through Exhibit AE in the Appendix. The reasoning of respondents who claim that prototyped systems are less expensive is undoubtedly related to their perceptions of time. Typical is the following:[18]

> Less cost, there's no doubt about it. It's very simple: time is money, and the quicker that you can see or anticipate problems and pinpoint them, the quicker they'll be identified and resolved and that means less cost.

The sophisticated understanding of the true system life cost found in the respondents' understanding of the time dimension is also seen here in the cost dimension. This understanding is typified by the following comment:[19]

> My reaction from a near-term cost perspective is that they [prototyped systems] are more expensive because development time goes up. From a long-term maintenance perspective, they will probably work out to be less expensive because there will be [fewer] changes that the system has to undergo after initial delivery.

Other respondents argue that costs are essentially the same. This reasoning is exemplified by the following quotation:[20]

> I would think it's the same cost, because the amount of time you're spending on the front end [prototyping] is about equal to the amount of time you spend at the back end, backtracking and adding [i.e., perfective maintenance]. So, it would be equal with us, in the long run.

TABLE 5.3

Cross-tabulation of time and cost dimensions

Time and cost dimensions

Cost dimension	Time dimension: Less time	More time	Same time	Undecided	Total
Less cost	15	1	2	1	19
More cost	0	5	1	0	6
Same cost	2	1	0	0	3
Depends	1	0	0	0	1
Total:	18	7	3	1	29

Others argue that prototyping is more expensive, although with other benefits to offer:[21]

> It may be more expensive to develop a prototype system than a traditional system... One of the biggest reasons is that the users tend to get more of the bells and whistles on the system because they have a lot more time to lobby for them, and they become a lot more aware of the value of an enhanced user interface...

We conclude this section by noting that, by virtue of the failure to assert either Hypothesis 2 or Hypothesis 3, systems developers have no direct time- or cost-motivation to use the prototyping method of systems development. We have demonstrated that prototyped systems are not delivered in significantly less calendar time or for significantly less cost than conventionally developed systems, in the perception of this sample of systems developers. There will be further discussion of these points in section 5.3.3.

This conclusion, however, begs a question: why, then, do systems developers who consciously adopt the prototyping method of systems development choose to prototype systems? One possibility is a perception on the part of systems developers that prototyped systems are of higher quality than systems developed in the conventional manner. This will be addressed in the next section.

5.2.4 Quality dimension

One of the last questions added to the interview instrument was a specific question regarding developers' perceptions of system quality. At the beginning of the research, it was felt that this could be inferred from commentary by the respondent. Later, it was decided to specif-

ically broach the question. This research interest is encapsulated in the following hypothesis, stated in alternate form:

Hypothesis 4: Developers perceive that prototyped systems are of greater quality than systems developed in the conventional manner.

Fourteen of the twenty-nine respondents received the question, though our report of the findings will be based on discussions with the full sample.

Eleven of the fourteen (79%) responses indicated that developers perceive prototyped systems to be of significantly higher quality than systems developed using conventional methods. The remainder (21%) perceived prototyped systems to be of equal quality. Employing the same parameter assumptions and level of significance as the previous two sections, this proportion's standardized score of 1.42 is significant at the 0.10 level of significance ($p < 0.0778$). Since small sample sizes mitigate against significance, this finding is particularly meaningful. Further, to maintain consistency, the assumed population proportion of 0.60 was employed. In this instance, an assumed population proportion of 0.50 is probably reasonable. Under this assumption, this finding is highly significant, with a standardized score of 2.14 ($p < 0.0162$). Figure 5.7 illustrates this finding.

Does the perception of quality reported here hold true across the classes of employment? As in previous sections, we cross-tabulated the type of employment of the respondents

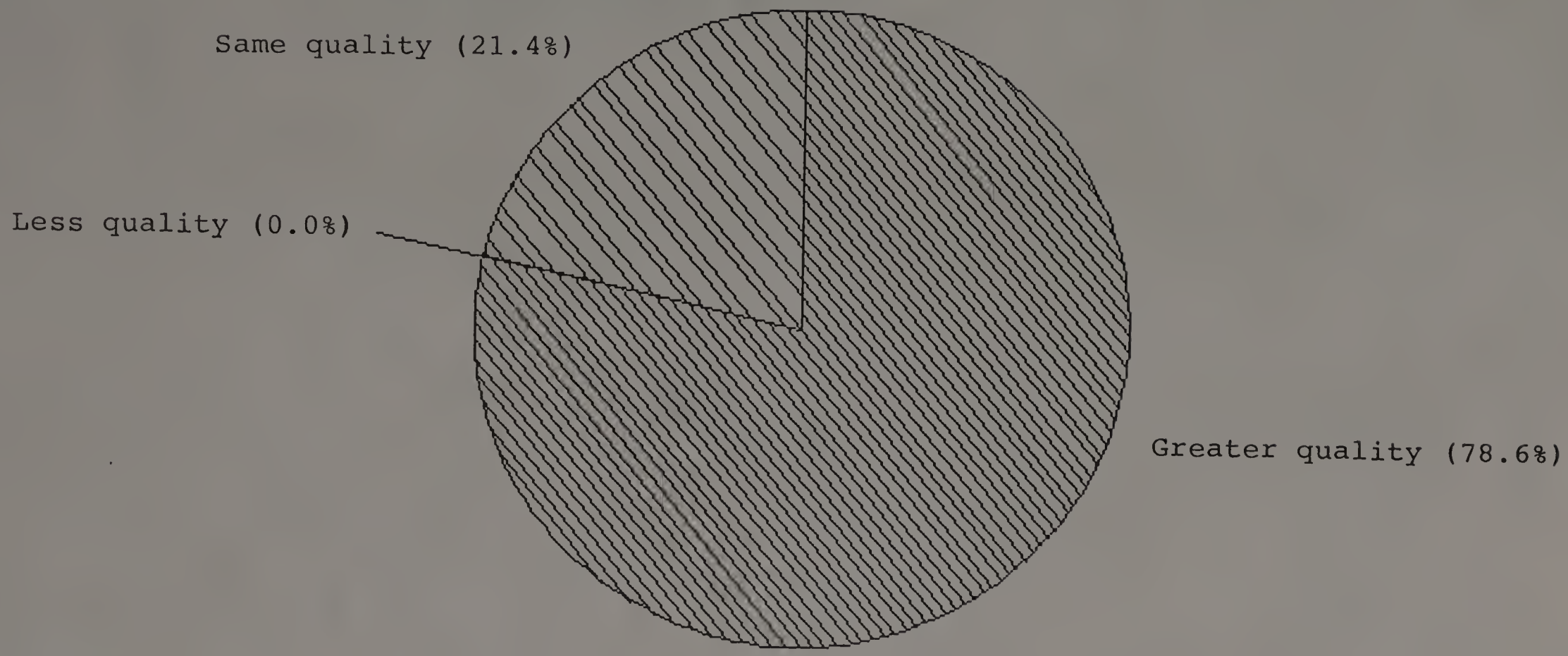


Figure 5.7

Quality dimension

(n = 14)

with their responses on the quality dimension. Seven of eight entrepreneurs (88%) reported a belief that prototyped systems are of higher quality than systems developed in the conventional manner. This proportion was significant ($p < 0.0001$). Four of six respondents employed by organizations concurred (67%), and this proportion was also significant ($p < 0.0401$). Both of these findings should not surprise us given that the sample as a whole generated a significant difference, as reported above. What is interesting is the fact that there is no significant difference between the groups ($p < 0.1711$). This leads us to conclude that the research population perceives that prototyped systems are of significantly greater quality, but that there is no significant variation on this dimension by type of employment. These results are presented in Table 5.2, previously presented.

This finding led us to undertake additional cross-tabulations. Recall that eleven of fourteen respondents who were asked a specific question regarding the relative quality of prototyped systems reported their perception that prototyped systems are of higher quality. Analyzing these eleven responses, we find that seven also believed that it was possible to deliver prototyped systems in less time, three believed that prototyped systems required more delivery time while one felt that there was no significant difference in delivery time. These proportions obviously differ, and our natural reaction would be to continue to the

next step: testing these sample difference to determine if they represent parameter differences. To do so, we would use the chi-square test of statistical independence, which is equivalent to the chi-square test for homogeneity of proportion across the subsamples. Unfortunately, the small sample size used forces a violation of a key assumption of the chi-square test. Thus, the proportions of the sub-samples reported in this paragraph is presented solely to gain insight and not as a statistical finding. These results are presented in Table 5.4.

In a similar fashion, we cross-tabulated the eleven respondents who felt that prototyped systems were of greater quality with their responses on the cost dimension. Seven of the eleven reported their perception that prototyped systems could be delivered at less cost, three felt that prototyped systems cost more, while one felt there was no significant difference between prototyped and conven-tionally-developed systems on the cost dimension. For the reasons explained in the previous paragraph, a chi-square test of statistical independence was not possible on these subsamples, and these results are presented for the sole purpose of gaining insight. These results are presented in Table 5.4.

As in the previous two sections, we will highlight the finding that the research population perceives prototyped systems to be of higher quality by presenting selected quo-tations from the interview transcripts. The transcribed

TABLE 5.4

Cross-tabulation of quality dimension
with time and cost dimensions

Quality and time dimensions:

Quality dimension	Time dimension: Less time	More time	Same time	Total
Equal quality	1	1	1	3
Greater quality	7	3	1	11
Total:	8	4	2	14

Quality and cost dimensions:

Quality dimension	Cost dimension: Less cost	More cost	Same cost	Total
Equal quality	2	1	0	3
Greater quality	7	3	1	1
Total:	9	4	1	14

answers to the specific question regarding quality are found in Exhibit C through Exhibit AE in the Appendix.

The following quotation is typical:[22]

> Since the user sees what they're getting, they're more involved in the design: it's their system ultimately... there's a qualitative difference if they just sign off on some piece of paper versus they know they participated in it from an early stage.

Similarly,[23]

> I think largely that the value of prototyping is that one allows the user to end up with a system that meets their needs better than a system developed using traditional methods. They [users] have a chance to control the direction of the development effort.

Other developers perceive the qualitative difference to be a function of the system itself, as the following quotation illustrates:[24]

> ...they're [prototyped systems] well thought out. There's a definite structure to them, there is a definite road map, the road map is a logical road map, and the pieces fit together quite nicely. And, the various hooks that are left that you need for future expansion are there, all in the proper places.

A minority of respondents felt that prototyped systems and conventionally developed systems were of equal quality. The following quotation is representative of this group of individuals:[25]

> Probably equal quality. ...From what I've seen in the systems that I've been involved in, people are willing to take ...[commercial] packages... and they're willing to adapt it to their uses. ...Whereas the prototype system, it fits them exactly... And, I suppose depending on the application that [prototyping] could be a better thing but I think the quality is pretty much equal.

The present researcher conjectured that some of the quality appeal of prototyping is that prototyping could better elicit user needs during the needs analysis phase of

systems development. To determine this, respondents were questioned on the efficacy of prototyping as a needs analysis tool. While a majority (seventeen of nineteen, or 59%) concurred that prototyping was more effective than conventional mean of eliciting user needs, this proportion was not significant. Eight of the twenty-nine respondents (28%) contended that circumstances determine the more effective tool, while four (15%) felt that the traditional needs analysis tools were superior, or that a combination of prototyping and traditional tools was called for. We can thus surmise that more efficacious needs analysis is not the motivating factor for using prototyping.

We submit that the findings presented in this and the previous two sections hold the key to prototyping's appeal for the developer community. The benefit of prototyping along the time and cost dimensions are unproven, both by this research and previously published research, although future research may demonstrate these points more clearly. What is new here is the quality perspective. Developers who consciously choose to prototype strongly perceive prototyping to deliver systems of better quality than systems developed conventionally. This key finding, we submit, is a major motivation for prototyping. We will expand upon this concept in some detail in the discussion portion of this chapter, section 5.3.3.

5.2.5 Conclusion

We have demonstrated the following four points in the research findings presented here:

1) There is substantial commonness of definition of the term "prototyping" among the research population of this study. It can be inferred that the respondent and the researcher were "on the same wavelength" in their discussion of the subject;

2) This study did not demonstrate that developers perceive that systems can be delivered in significantly less calendar time than systems developed in the conventional manner. Note, however, that there remains enough uncertainty on this point to warrant continued research;

3) This study did not demonstrate that developers perceive that systems can be delivered at significantly less cost than systems developed in the conventional manner. Like the time dimension, there exists motivation for continued research in this area; and

4) The most important finding of this study was that developers perceive that prototyped systems are of significantly higher quality than systems developed in the conventional manner.

Taken together, these findings present a coherent picture of developers' perceptions: developers adopt the prototyping approach in order to deliver high quality systems, despite weak indicators (at best) that their choice of development technique allows them to deliver systems in a

more timely or more cost-effective manner. As one respondent noted:[26]

> ...[Prototyping] is the right way of bringing experts in... to get the correct product, the product that will actually do what we need it to do. And, by and large, people aren't willing to pay for better products. Prototyping to me is not aimed at being a more cost-effective or a more tightly disciplined approach, it's aimed at producing a better product...

or, even more succinctly:[26]

> Obviously, I think [prototyping] is the best way to go or I wouldn't be doing it.

It is most reasonable to expect that quality is indeed a major motivator for developers to choose prototyping as a development method. Yet, we would be naive to think that quality itself is the sole motivator for the use of prototyping. We submit that the desire to deliver systems of higher quality is part of a variety of motivating factors driving developers in the research population to choose prototyping. The following section of this chapter is a general discussion that will explore this variety.

5.3 Discussion

Beyond the motivation of delivering high quality systems, what other factors motivate developers to adopt the prototyping approach to systems development? It should be recalled that a large number of respondents are full-time entrepreneurs (see Figure 5.1), while others hold positions in which the efficient and economical delivery of systems undoubtedly is a key factor in their success. This is likely to be representative of the research population. As noted previously, external indicators point clearly to the

affluence of the research population. It seems likely, therefore, that there exist some economic motivations for the use of prototyping. The first portion of the discussion addresses these.

5.3.1 Direct economic motivators

Several classes of direct economic motivators can be envisioned. In this section, we will examine two: whether prototyping allows the developer to derive additional income from the project; and whether prototyping impacts the financial operations of the business, particularly cash flow.

No respondent indicated that prototyping was chosen to maximize income from the project s/he is currently working on. Indeed, several respondents brought up concerns regarding the (at least superficially) unstructured nature of the prototyping approach. For example:[28]

> ...I've quoted and have done job proposals based upon time I've estimated and I have grossly underestimated the time... prototyping does tend to exaggerate the time frame. I don't think you can adequately charge for all the time you have to put in, too, because otherwise the system becomes too expensive...

Why, then, does this respondent choose to prototype?

> I think [prototyping] has cost us financially. However, on the other hand, prototyping has benefited us in client loyalty. I have no doubt about that.

This respondent was speaking from the perspective of a fixed-price contract for a project. Others, having been burnt by this arrangement, refuse to develop under those conditions. Note this response:[29]

> ...a fixed price basis. That's not the way we work and we will reject contracts that say that... it's a loaded gun at the developer's head. I'd rather be as honest as I can about what I think it's going to cost, but document clearly everything we are doing versus everything that was originally asked for, so that if there are cost overruns [clients] see very clearly that it's tied to things they requested.

Other respondents feel even more strongly on this issue:[30]

> In cases where we don't think we can prototype, we turn down the work, so we lose work. I mean, it actually costs us money because we require that our clients work with us in the way we want to work. In fact, I've turned down incredible amounts of work...

This researcher feels confident that prototyping is not viewed as a means of maximizing income for the present project.

Does prototyping have a relationship with the ongoing financial operations of the firm?[31] Only 15% of the respondents receiving a direct question on this point claim that prototyping has a substantial impact on their financial operations. Larger proportions felt prototyping had a moderate impact (46%) or a minimal impact (39%). Figure 5.8 illustrates these findings.

Few respondents felt that prototyping substantially impacted financial operations, and those that did took a very practical viewpoint:[32]

> I would say it's a very strong relationship... It's not a direct connection... if we pick the wrong tool, we end up wasting a lot of money and have to eat that time because the client's not going to pay for it. So, I would say it's a medium-strong relationship.

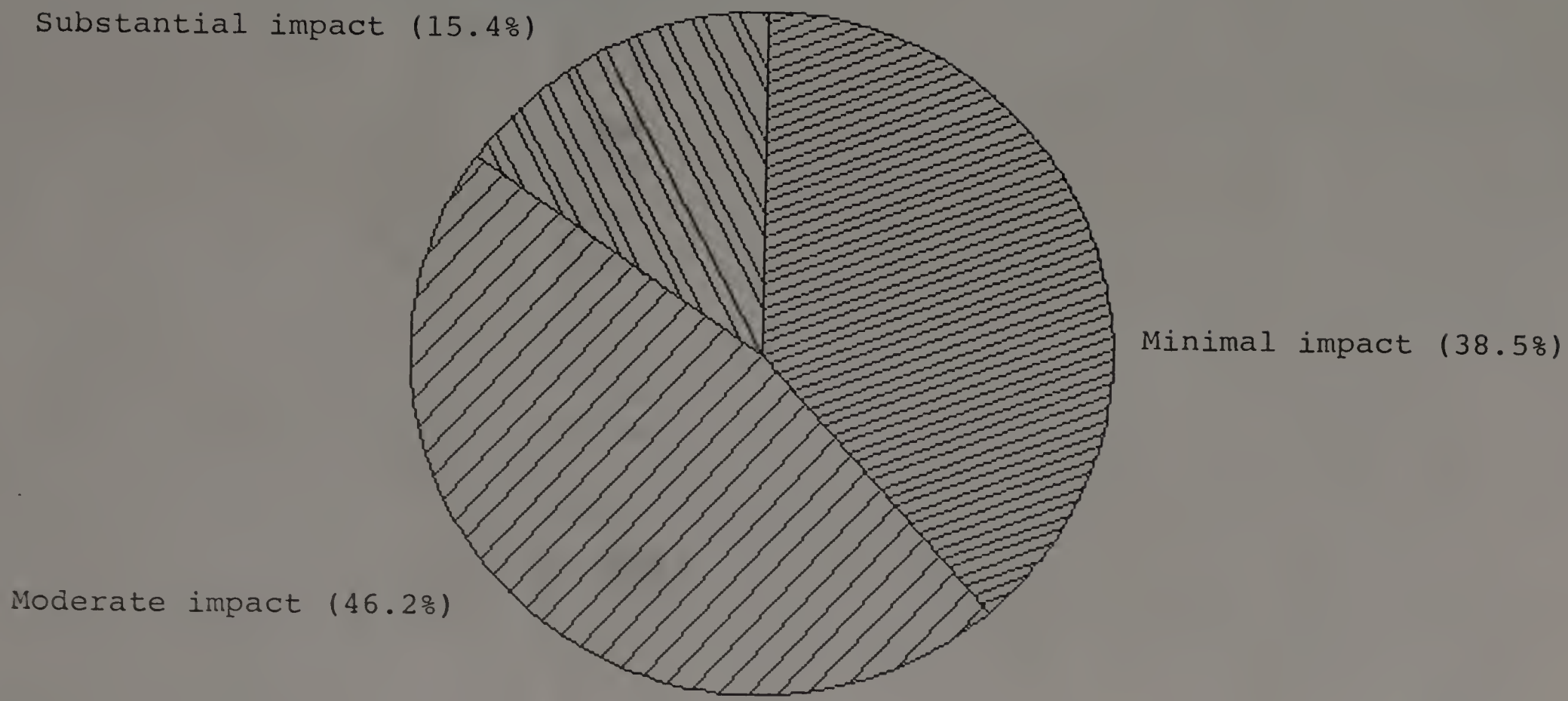


Figure 5.8

Financial impact of prototyping

(n = 13)

Whereas, a more mainstream view is this:[33]

> [Prototyping] helps [business operations]. You get a clearer picture of what the project's going to be. And we find we can bill people for the prototype, because it's a value added to them... it's not like you have to build something first and then discuss what the ultimate cost is going to be, because you can... bill them hourly. So, it actually helps a lot.

Other respondents saw less of a direct relationship:[34]

> I think by being very good at prototyping, we develop better systems for our clients, and the better job you do the more money you make... it's a two-step relationship, that prototyping improves the quality of the work and the quality of the work produces the financial conditions of the company.

Those respondents who perceive prototyping to be primarily a marketing device also see an indirect relationship:[35]

> ...the technique of prototyping has allowed us to actually produce a marketable product far sooner than we would have before. And, in marketing that product, we've generated an income which has paid for the development...

This respondent clearly agrees with the strategic perspective articulated by Ives and Learmonth [Ives and Learmonth, 1984]

Finally, other respondents see no financial linkage between their choice of the prototyping method and their financial condition. Typical of this type of response is the following, which emphasizes the general value of the technique:[36]

> I think the impact is a more professional appearance for our organization, that we really know what we're doing as far as design and analysis. It promotes us as an organization that's not trying to blow something by a client but instead is giving [the client] the chance to say 'yea' or 'nay' to it. It's a marketing tool.

We must conclude that there exist no direct economic motivators for developers to choose the prototyping method

of systems development. There undoubtedly exist other, indirect business-related motivators, beyond the purely economic. One that came through clearly in discussion with respondents is the degree to which respondents are dependent on repeat and referral business for their continued financial well-being. For example, a number of respondents reported pridefully that they have never advertised for business, but rely solely on word-of-mouth to generate business. This aspect of the respondents' business affairs was beyond the scope of this research, but it does give rise to future research hypotheses. This topic will be discussed in some detail in section 5.3.3. In the next section, we will explore other potential motivators for prototyping.

5.3.2 Other motivators

We have seen that direct financial considerations are weak motivators for prototyping. What other factors would motivate a developer to adopt the prototyping technique? In this section, we will discuss four possible motivating factors: (1) a reduction in developer anxiety; (2) an increase in positive relationships with clients, resulting in (3) a heightened sense of collegiality felt when prototyping; and (4) an increase in personal satisfaction.

The interview questionnaire (Exhibit A in the Appendix) contained a series of questions addressing the developer's degree of comfort with the development process when prototyping. The results of this questioning make it clear that prototyping reduces the anxiety felt by developers. Noting

that their income derives primarily from systems development (recall that eighteen of twenty-nine respondents are full-time entrepreneurs), a reduction in anxiety is a powerful motivator indeed.

Twenty-seven of the twenty-nine respondents dealt with questions that addressed developer anxiety. Twenty-one (78%) reported that they felt less anxiety when prototyping, while four (15%) reported higher anxiety levels. One reported the same degree of anxiety, and one was unsure. These findings are illustrated in Figure 5.9.

These findings can be highlighted by examining responses to this question. Typical of those reporting less anxiety is this respondent:[37]

> ... I feel I've gotten more control over the project because I don't have [a] predefined, inflexible set of rules to develop by. Nothing's worse than you have to do a project and find out that [the] original definition is wrong, but it's cast in stone and you can't get it fixed.

Further, argues one respondent, prototyping corresponds to the mind set of systems developers:[38]

> ...you want fast feedback, that's what people whose brains work that way thrive on, that's why people become hackers and get engrossed in computers, they want rapid feedback. And I find it very anxiety-provoking to have to stop and walk away from the rapid feedback environment and sit down and write reports, papers and design documents and so on...

This respondent probably summed up the reaction to this question the best:[39]

> There's a lot less fear of being wrong in a prototyping environment.

A minority viewpoint was that the prototyping development environment leads to greater anxiety. For example:[40]

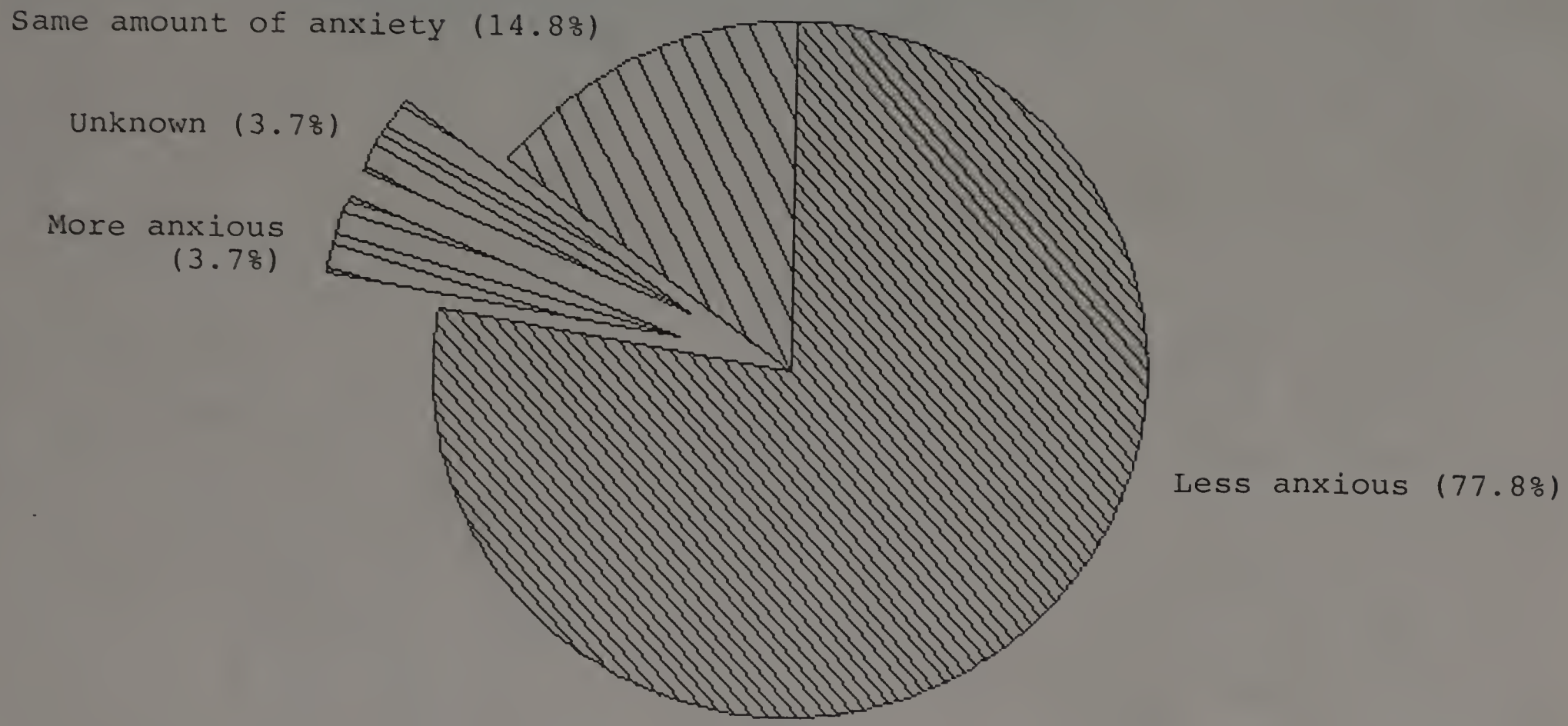


Figure 5.9

Degree of anxiety when prototyping

(n = 28)

> The truth is more anxious... It should be less because I know the end product will be better. The reason it's more is because I know the costs will be higher. When I'm involved in prototyping, I'm constantly scared that they're [users] not going to understand why the cost is going up, despite the improvements I'm making, and therefore I'm working harder to hide costs, to cover it at lower costs, to put in that extra time that they're not billed for, to do everything I can to try to keep the costs as low as possible, and that puts inordinate pressure on me...

Still other respondents find the prototyping process itself a source of anxiety:[41]

> Probably more anxious. I'm always nervous dealing with a new project because you are dealing with unknowns, as far as how the clients will react to this prototyping process and things like that. And, it's kind of difficult to gauge until you're actually in there how they will react to the prototype as opposed to you just dropped a package in.

We submit that the perception by developers of a reduction in anxiety represents an important non-economic motivator for prototyping. The reaction of clients to the prototype as well as the prototyping process was a concern expressed by a number of respondents. An additional fruitful area of interest is a possible change in the relationship between developer and client. This will be addressed in the following paragraphs.

All respondents received several questions regarding their role vis-á-vis clients in the consulting process. Of particular interest was the developers' perception of the leadership role they must assume when prototyping. Would the role of project leader, commonly delegated to the computing professional in conventional development environments, remain with the developer when prototyping? Somewhat

to our surprise, the sample reported they must take on a leadership role more often when prototyping than in conventional development. Of the twenty-six respondents who addressed this question, thirteen (50%) reported that they needed to take on a leadership role more often than in conventional development, while eight (31%) reported that they take on leadership less often. Three indicated that the degree of leadership is the same in both forms of development, while an additional two indicated that the leadership role depends on factors not covered by the question. These findings are illustrated by Figure 5.10.

Discussion confirmed the interviewer's impression that this leadership role pleased the developer. It is, in fact, the major cause of the anxiety reduction reported. The respondents gave some insights into the leadership process when prototyping. A sampling of their reaction follows:[42]

> Prototyping should result in a smaller development team, therefore the Information Systems person should be providing the leadership role, the drive behind the execution... The users have not been involved [in the past]. So, you can't just one day flip a switch [and have users] take over responsibility for all these things... The IS [person] has to take on the leadership role to ... sponsor getting the user involved until they start taking it out of our hands...

In part, this is because the systems developer bears the onus for the success of the system, as illustrated by this quotation:[43]

> ...I am ultimately responsible for the success of the [system being developed]. I always feel that I have to be responsible but that also means I must responsibly listen to what the client's desires are...

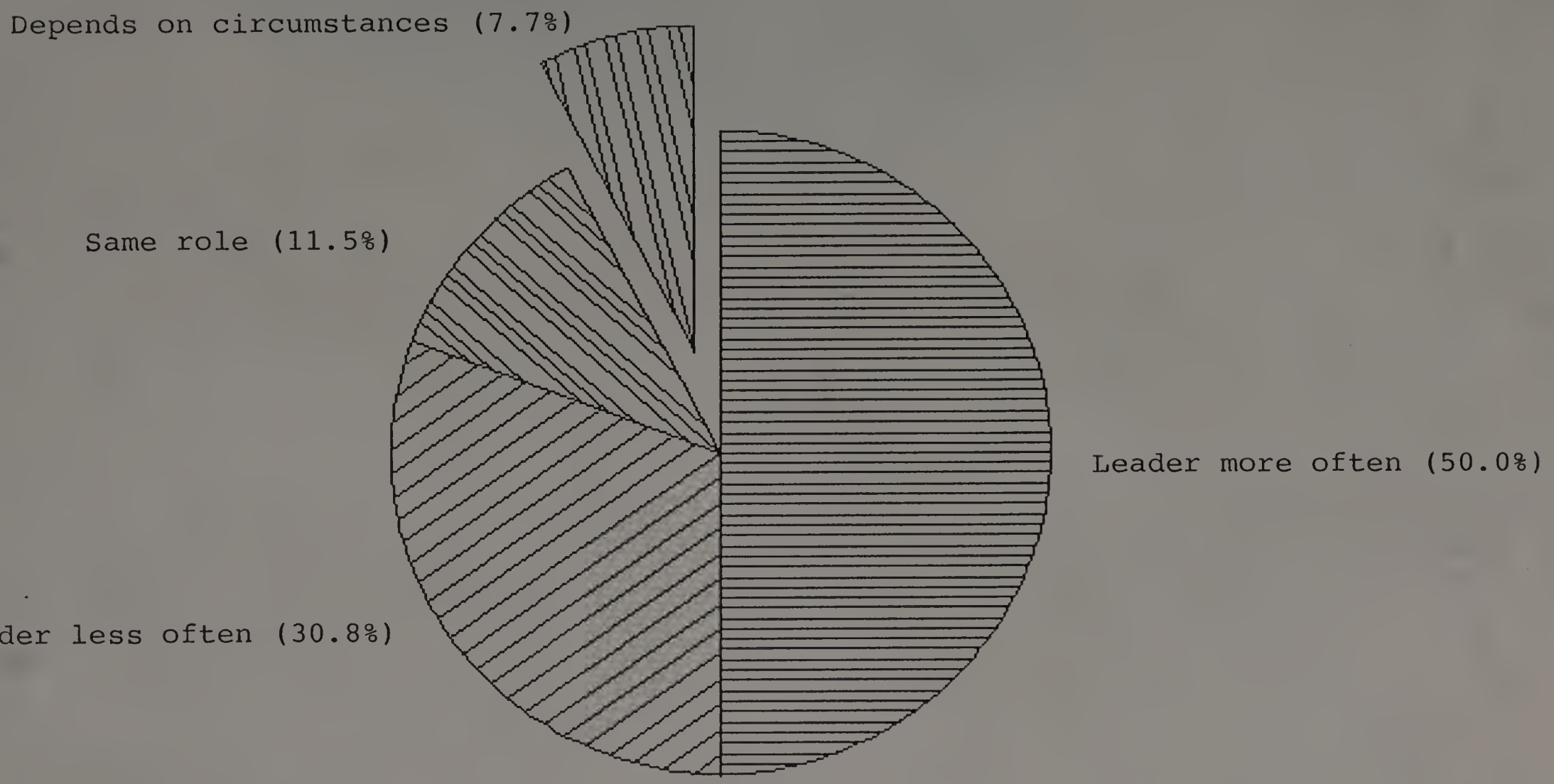


Figure 5.10

Perception of leadership role

(n = 26)

Along the same lines, the need to bring out client needs was emphasized by another respondent:[44]

> Sometimes clients are really reluctant to bring up things because they think they're too minor a thing to discuss. You have to draw things out of people as to what they would like...

The minority position, that the developer takes on a leadership role less often, is encapsulated by the following statement:[45]

> It strikes me that I would probably be less of a leadership role... I try to let the clients who really know what they're doing take a leadership role and I try to follow what they're trying to do.

Still others see the leadership role as a function of the business relationship, that is, leadership depends on the circumstances of the development, rather than the development environment:[46]

> It's the same. As long as you are up front to the client, what are the steps going to be, and the client knows that the first stage is a prototype, it's not the working application, and we're going to be going back and forth for a while, that's fine.

As a result of these findings, we desired to seek what specific form "leadership" was taking. To this end, twenty-eight of the twenty-nine respondents addressed a question regarding whether or not users bring up sufficient system options on their own, or whether the developer must take the lead in presenting system options. The findings of the leadership question where confirmed, as sixteen of the twenty-eight (57%) reported that the developer had to take on presenting system options more often when prototyping than when engaged in conventional systems development. Five (18%) felt the role the developer played was the same as in

conventional development, while four (14%) felt the options-presentation issue was a function of factors not included in the question; only three (11%) felt the user took the lead in presenting sufficient system options. Figure 5.11 illustrates these findings.

A typical comment from a respondent on this issue is:[47]

> It's funny: as well as they usually know their businesses, they aren't necessarily logicians. I find I tend to be the one who brings up alternatives and in many cases changing the way they perform some task even manually.

This respondent carries this argument to a logical conclusion:[48]

> Clients want us to... [If users bring up options], it's only for a short while until they get to know us. If they were at that level [users bringing up system options], they'd be doing it [development] themselves.

Some respondents report a change in perception by clients:[49]

> I think initially clients want me to take the lead. They sit down and they're looking for all the answers. But, after... a while, they really get into it... and tell me what I need to do.

Still others perceive an equal sharing of the task of working through system options. For example:[50]

> I would say it's about 50-50. Some clients, you suggest things and they say 'Gee, I never thought of that', and some people are just filled with ideas themselves. It really just depends on the people's personal backgrounds, what their exposure to computers is and what their exposure is to the field that is being worked on.

Still others perceive a stronger client role on this issue:[51]

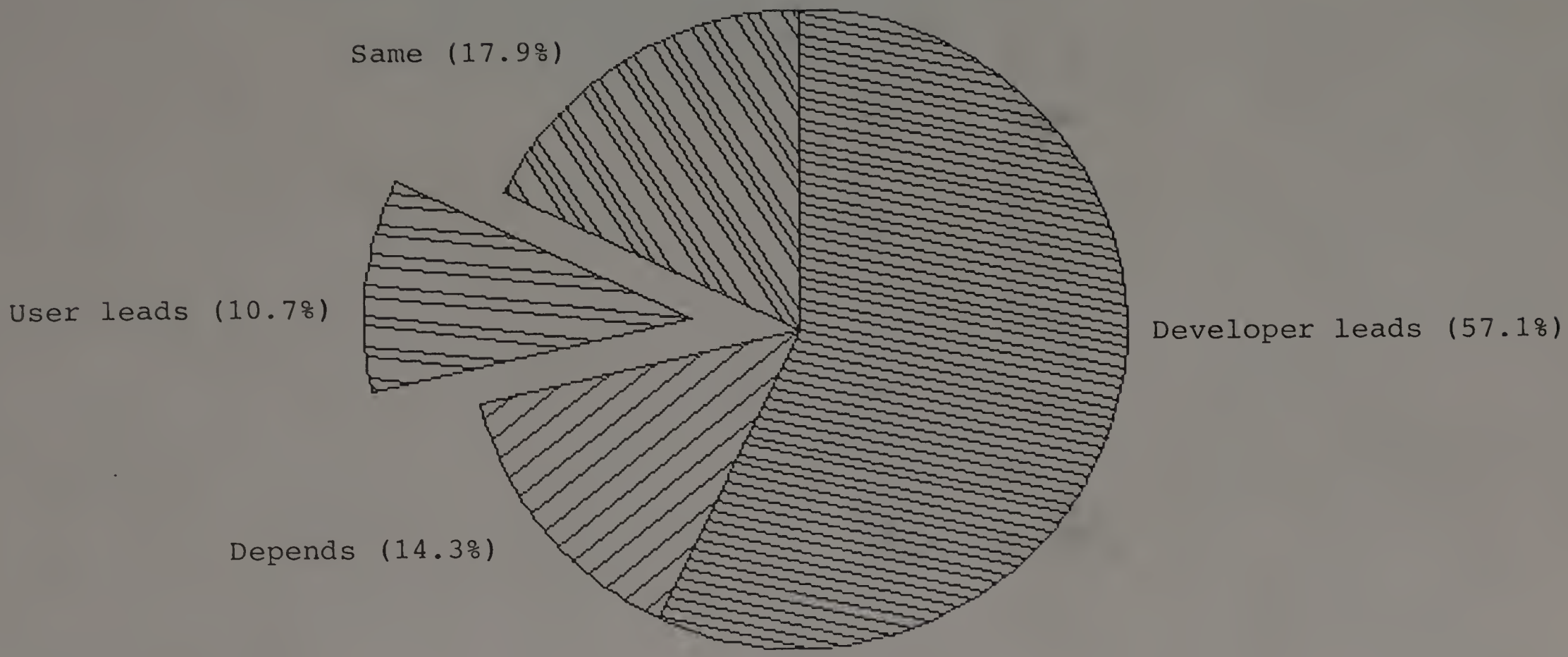


Figure 5.11

Presentation of systems options

(n = 28)

> Our users are fairly aggressive and generally not hesitant to come up with new ideas. I come up with ideas myself and I'll take the user role there and go to one or two of them [with systems options proposals]... It is, at this point, fairly well balanced.

We further submit that the issue of options-presentation discussed above is characteristic of the developers' perception of a heightened sense of professionalism. In part, this may be a reaction to the former role of the systems developer. In the former role, the systems developer was in a service role, but his/her control of a very valuable resource placed him/her in a role called a "priesthood" by one respondent:[52]

> ...there's a group of experts out there [conventional analysts and designers that employ] the guru approach. I don't believe [in] that. I think that's a mentality that is... old-fashioned. It's a mentality that I think most computer people had ten years ago, and maybe it was true ten years ago. But, more and more users are getting involved in systems design anyway, especially with the advent of microcomputers. And, I think users know what they want... So, the idea that there is a priesthood out there that really knows what's going on... I know that's not true.

The following respondent was even more succinct:[53]

> I think it's healthy, because it's a trend toward group participation and a little more democracy in what has been a ivory tower... activity. And ivory towers usually deserve to be pulled down, and certainly this one does.

It appears that respondents who choose to prototype are strongly motivated by the relationship they develop with clients. Note that the research population is defined to be independent developers who utilize prototyping. It is not possible to perform a causal analysis here: does prototyping generate this professional relationship or does prototyping spring naturally from a professional relationship that

emphasizes collegiality? Such a question must wait on future research efforts, although past research efforts are relevant [Dagwell and Weber, 1983]. However, it is clear that these respondents, and by extension the research population, clearly perceive themselves to be colleagues, and not "priests" or "gurus".

One motivating factor not yet discussed is personal satisfaction. Beyond purely economic factors, it is clear that some respondents are attracted by the job itself. It is clear that the skills possessed by the respondents are highly valued in the marketplace. It is likely that any respondent could easily acquire a conventional job whenever desired. Part of the attraction of the independent developer's role is the intellectual challenge. For example:[54]

> I guess one thing that characterizes me as a programmer and system developer is that I really regard this as play and as fun. I get involved in my programming as I would if I were doing a hobby... What attracts me about this kind of work is the problem-solving... it seems to me to be intellectual play. That's what I like about it and that's one reason why the prototype method appeals to me. It's... creating toys that behave like real systems and then turning the toys into real systems. And that seems fun to me...

Similarly, one respondent stated,[55] "I <u>enjoy</u> doing it; it's less a job than a labor of love" (emphasis in original). Clearly, therefore, the personal satisfaction of developing systems motivates developers to choose prototyping as a development methodology. This sense of personal satisfaction is not unrelated to the economic well-being of the developer. Undoubtedly, the developer who feels fulfilled

projects a sense of confidence, leading one respondent to state:[56]

> ...prototyping... does not seem to be the factor that drives companies to have software written for them. The thing that drives them is the assurance by somebody like me who they have either heard is good or they've had experience with ...that's what they're looking for.

In short, our discussions with the respondents indicate that there exist strong non-economic motivators for prototyping. Collectively, these non-economic factors represent an improved working environment that seems to strongly attract the research population to prototyping.

5.3.3. Interpretation

Our contention is that independent MIS application software developers choose prototyping for a variety of reasons. Primary among these are a desire for high quality systems and an improvement in their working environment. This research has not indicated, however, that more timely or more cost-effective production of systems motivate these developers to choose prototyping, nor do direct economic factors provide a motivation to choose prototyping.

We contend, however, that there exists a substantial economic motivator not brought out thus far in this report. This research has shown that the quality of the system delivered is a strong motivator for the use of the prototyping method of systems development. Further discussion in section 5.3.1 establishes that other direct economic factors are not strong motivators. We conjecture that other factors are stronger motivators: specifically, we hint that a heightened sense of their collegial role, together with a

distinct reduction in anxiety and a heightened sense of self-satisfaction, might be key motivators for systems developers to utilize prototyping. As these findings go beyond the research hypotheses of this project, and, in fact, were generated in the search for answers to those research hypotheses, they remain conjecture at this point. These findings do point the way toward additional research in the prototyping arena, however.

To come to an understanding of how prototyping does indeed provide an economic benefit, we must understand the milieu in which the developer finds him/herself.

As noted earlier, these individuals are self-directed, whether as entrepreneurs or as autonomous individuals within organizations. Their livelihood depends on a steady stream of incoming assignments. Further, note that none of the respondents advertise, relying instead on word-of-mouth referrals for new business, as well as repeat business from existing clients. A displeased client would certainly poison this environment; thus, developers have an economic incentive to deliver systems that clients perceive to be of high quality and reasonably priced.[57]

How does prototyping contribute to this situation? Understanding this situation gives us an insight as to why developers will sacrifice time and cost benefits in order to prototype. This researcher interprets, however, that any time- or cost-disadvantage is transient. In fact, the main focus of this interpretation is why the perception of the

developers is actually incorrect.

First, there is the issue of prototype generating ongoing business for the client. The following extended quote is typical of the comments the researcher encountered:[58]

> I think financial considerations are the key to prototyping... I firmly believe that what makes me personally happy is a happy client and that's foremost. The second thing is to have the client using what you have developed... I've heard so many stories... where they developed a nice system and the client thought it was a nice system but they never used it for one reason or another. And, I find that doesn't happen with this iterative prototyping cycle, that it gets the client very much involved, it develops some very strong personal bonds. It helps your business... because now... our original client has worked with us and gotten to know us. ...in our company, we don't have a typical salesman type that comes in, very well dressed ...and glad-hands and everybody swoons... we don't have people like that. We have very smart people that don't necessarily have the great social graces... it's difficult for us to get the initial foot in the door because we don't have that great first image. But, what is incredible to me is all the follow-up business we end up getting, because once clients start working with us... I think the personal contact is so important even for marketing.

Clearly, personal satisfaction, a desire for quality systems and the economic incentive of additional business are combined in the perception of this respondent.

Second, the key to understanding the indirect economic motivation for prototyping is the reuse of components of one system in a subsequent assignment. Indeed, the concept of a "software library" is an old one in the computing field. In this survey, twenty-five of twenty-eight (89%) respondents reported using database management systems in their development efforts. Discussion revealed that the respondents especially relied upon the ancillary tools associated with

these products. A characteristic of these ancillary tools is the easy reuse of modules from previous development experiences in a new undertaking. Further, it is not unusual to use the data-manipulating features of the DBMS itself to catalog modules from previous projects.

In addition, nineteen of twenty-seven (70%) of respondents indicated a conscious reuse previous systems' components in new development, while another two (11%) noted that this is possible under some circumstances. Thus, we see a regular pattern of reusing system modules across several systems[59]. Even those few respondents who did not indicate reusing components (five of twenty-seven, or 19%) undoubtedly rely on the experience and knowledge gained in one development experience to make future assignments more productive.

If we combine the motivations for quality in its own right with the large amount of reuse of system components, we arrive at a major economic incentive for the use of prototyping. First, the present client must be satisfied in order to garner new business from that client or others referred by that client. Second, the developer him/herself has significant incentive to produce a high quality system since the components of that system will be used in future systems. Accuracy and quality can be so important that they overwhelm consideration of time and cost factors.[60] When that reuse occurs, the developer will want to reuse those components that have stood a major test: they have been

examined by clients and found to be successful. This, we submit, is an underlying economic motivation for prototyping not made directly clear by the research reported here.

In the following, concluding chapter, we will summarize these results and point out the implications of these results to the various actors in the information systems arena.

Notes

1. See the comments of Respondent 2 in Exhibit D.

2. See comments of Respondent 18 in Exhibit T.

3. See comments of Respondent 28 in Exhibit AD.

4. See comments of Respondent 11 in Exhibit M.

5. See comments of Respondent 26 in Exhibit AB. Respondent 1 would agree with Respondent 26: see Respondent 1's comments in Exhibit C.

6. See comments of Respondent 13 in Exhibit O.

7. See comments of Respondent 21 in Exhibit W.

8. See comments of Respondent 27 in Exhibit AC.

9. Recall that one respondent (Respondent 19) actually had no prototyping experience.

10. The author wishes to thank Committee Member Craig Moore for suggesting this line of inquiry.

11. The author wishes to thank Committee Member David Stemple for suggesting cross-tabulation as an analysis approach here and later in the chapter.

12. See comments of Respondent 3 in Exhibit E.

13. See comments of Respondent 16 in Exhibit R.

14. See comments of Respondent 21 in Exhibit W.

15. See comments of Respondent 1 in Exhibit C.

16. See comments of Respondent 26 in Exhibit AB.

17. See comments of Respondent 20 in Exhibit V.

18. See comments of Respondent 18 in Exhibit T.

19. See comments of Respondent 1 in Exhibit C.

20. See comments of Respondent 25 in Exhibit AA.

21. See comments of Respondent 26 in Exhibit AB.

22. See comments of Respondent 18 in Exhibit T.

23. See comments of Respondent 26 in Exhibit AB.

24. See comments of Respondent 25 in Exhibit AA.

25. See comments of Respondent 24 in Exhibit Z.

26. See comments of Respondent 8 in Exhibit J.

27. See comments of Respondent 7 in Exhibit I.

28. See comments of Respondent 9 in Exhibit K.

29. See comments of Respondent 4 in Exhibit F.

30. See comments of Respondent 13 in Exhibit O.

31. This question was generated by early responses during the pilot study phase. It was added to the questionnaire seen in Exhibit A relatively late in the survey. As a result, the question was explicitly asked of only fourteen respondents, one of whom has no prototyping experience. For this reason, n = 13 on this response. As this data is presented in discussion only, this is felt to be a sufficient base for inference.

32. See comments of Respondent 25 in Exhibit AA.

33. See comments of Respondent 18 in Exhibit T.

34. See comments of Respondent 26 in Exhibit AB.

35. See comments of Respondent 15 in Exhibit Q.

36. See comments of Respondent 6 in Exhibit H.

37. See comments of Respondent 15 in Exhibit Q.

38. See comments of Respondent 21 in Exhibit W.

39. See comments of Respondent 26 in Exhibit AB.

40. See comments of Respondent 8 in Exhibit J.

41. See comments of Respondent 24 in Exhibit Z.

42. See comments of Respondent 5 in Exhibit G.

43. See comments of Respondent 17 in Exhibit S.

44. See comments of Respondent 24 in Exhibit Z.

45. See comments of Respondent 23 in Exhibit Y.

46. See comments of Respondent 25 in Exhibit AA.

47. See comments of Respondent 7 in Exhibit I.

48. See comments of Respondent 11 in Exhibit M.

49. See comments of Respondent 23 in Exhibit Y.

50. See comments of Respondent 24 in Exhibit Z.

51. See comments of Respondent 15 in Exhibit Q.

52. See comments by Respondent 18 in Exhibit T.

53. See comments by Respondent 16 in Exhibit R.

54. See comments of Respondent 20 in Exhibit V.

55. See comments of Respondent 11 in Exhibit M.

56. See comments of Respondent 1 in Exhibit C.

57. The author is grateful to committee member Craig Moore for this insight.

58. See comments of Respondent 17 in Exhibit S.

59. The author is grateful to committee chair Van Court Hare, Jr., for suggesting this insight.

60. The author is grateful to committee member Conrad Wogrin for this insight.

CHAPTER 6

CONCLUSION

This chapter opens with a brief reiteration of the major findings of this research. We then proceed to a discussion of the implications of these findings on major actors in the systems development process: the developers themselves, clients and educators. The chapter concludes with a statement on the limitations of the present research findings and a discussion of future research needs.

6.1 Reiteration of research findings

For clarity and to provide a sense of closure, we restate here three major findings of the present research. The complete report on the research findings is presented in the preceding chapter.

The most substantial finding from this research is that the quality of the prototyped systems is a substantial incentive for independent systems developers to choose prototyping. Quality is desired both for its own intrinsic value and, as we have seen in discussion, for the indirect economic benefits it provides. Note that our research concludes that the prevailing assumptions about prototyping's cost- or time-benefits are unfounded.

Prototyping also provides a pleasant work environment. Anxiety is reduced and a more collegial relationship with

the client is established, leading to greater enjoyment of the job by developers. Since these developers have chosen this work style in a high-demand employment situation, it is reasonable to conclude that improvement in the work environment is a strong motivator to choose prototyping.

Finally, although we have established by this research that developers perceive no direct time- or cost-benefit in prototyping, we interpret that economic motivations to prototype do, in fact, exist. Our position is based on the report by a very large majority of developers that components from one prototyped system are reused in subsequent development efforts. Thus, developers are able to generate future systems based on components that have already received a favorable reaction from previous clients. At the very least, the developer can take the knowledge and sensitivity gained from intensive interaction with a past client and apply it to a future client, even if no specific reuse of components takes place. Thus, developers are willing to sacrifice time- and cost-benefits when prototyping an original development, because they intuitively know the high-quality components derived will serve them well in the future, thus providing an indirect economic motivation for prototyping.

Taken together these three factors form a compelling set of incentives for developers to select the prototyping method of systems development. In turn, these motivations are based on a strong intuitive sense of the true life cycle cost of systems, particularly the high cost associated with

perfective maintenance of established systems. By using components whose quality has been verified by past clients, in a personally-satisfying work environment, the independent developer achieves a high level of professional self-actualization.

6.2 Implications for systems development actors

We now address the implications these findings have for actors involved in the systems development process. We address three classes of individuals: the systems developers themselves; clients of systems developers; and educators involved in training future systems professionals. Given the research focus of this dissertation, most of our attention will be paid to the first class of actors.

6.2.1 Systems developers

This research has indicated that systems developers do not perceive time- or cost-benefits in prototyping. Nonetheless, the flexibility and responsiveness of the prototyping environment make it easy to promise too much along these dimensions. Developers should take care not to "sell" prototyping as a quicker or more cost-effective development method. On the contrary, developers should try consciously to dampen time- and cost-related expectations.

What developers should emphasize is the quality dimension. This research clearly indicated that developers perceive prototyped systems to be of increased quality. From the client's perspective, "quality" is often seen to be the degree to which the system conforms to the client's expec-

tations. This is the point on which developers should "sell" prototyping: when prototyping, the developer has much more assurance that the system derived will conform to the user's expectations and view of the application.

At present, not enough is known about the prototyping environment to allow developers to bid on fixed-price contracts (see the discussion in section 6.4.2, below). This is because it is very difficult to predict the number of iterations it will take to satisfy the client. Therefore, all bid procedures that mandate a fixed price before the contract is let should be avoided by developers who wish to prototype. This restriction eliminates a large segment of potential business. In particular, government agencies often require both detailed specifications documents and fixed-price contracts. Thus, a major implication of the choice to prototype is the opportunity cost of losing access to a significant segment of the systems development business.

Beyond the loss of potential business when fixed-price contracts are called for, developers must educate clients on the true cost of a system across its life cycle. All too often, a client perceives a point at which a system is "finished", and wants a fixed-price contract to achieve that point. Clients need to be educated about the phenomena of perfective maintenance. Prototyping offers the promise of reducing the perfective maintenance component of the true cost of a system over its life cycle. Clients need to be informed of this, and achieve the same level of intuitive

understanding of this point that developers apparently have achieved.

This research focused exclusively on the development of application software within the domain of Management Information Systems. Developers should be wary of applying the findings of this research to other domains of activity. For example, the finding that prototyping achieves a greater conformity to the user's expectation, thus reducing perfective maintenance, does not necessarily hold true in a computationally-intensive scientific system. In such a case, a great deal of development effort is directed at the underlying algorithms, and far less effort is directed at the points where the system interacts with the user. Moreover, in most cases, the user of such a system is computer-literate and can adapt easily to any interface presented. Prototyping could be of questionable value in such a development.

Perhaps the most significant implication of this research is the indirect economic benefit of prototyping. If our interpretation is correct, prototyping provides long-term economic benefits to the developer, as s/he builds a library of client-tested modules to apply to future systems. The developer must therefore be prepared to place substantial effort into the development of each module in turn, thus diminishing any time- or cost-benefit of prototyping even more, in order to realize this long-term benefit. To balance this with the rushed and deadline-filled world of

systems development requires both resolution and a deep understanding of the long-term objective.

Finally, it must be recognized that the iterative nature of prototyping implies a highly structured system design. Prototyping is not an excuse for chaos: developers must be able to pinpoint exactly where the code that is causing concern to users is located. Having altered that code, the developer must be sure there will be no unintended "ripple effects" elsewhere in the system. These factors call for a strong and well-designed structure underlying the system.

6.2.2 Clients

Clients want systems that conform to their expectations, and experience has shown that conventional development often fails along this dimension. Clients need to understand that prototyping holds much promise for addressing this inadequacy of conventional design. However, prototyping implies certain other factors, which must be considered. The most important of these is its iterative nature. At our present level of knowledge, we cannot predict with assurance how many iterations will be required to achieve client satisfaction. Therefore, clients would be wise to consider alternative development methods when faced with serious time or cost constraints, even if these alternative methods result in a system that does not conform to their expectations as well as a prototyped system would.

Clients also need to become more sophisticated about the true cost of the system life cycle. In particular, they

must become aware that "bugs" are, more often than not, a need for the system to adapt to their expectations rather than an outright error. A "finished" system that does not satisfy users is not, in fact, "finished". A system with a protracted period of perfective maintenance is, in fact, a prototyping exercise drawn out over a long period of time.

These findings should cause clients to question the appropriateness of detailed specifications and fixed-price contracts, particularly for systems heavily dependent on user interfaces.

6.2.3 Educators

Educators must prepare both developers and clients for their respective roles in the systems development process. Clients can best be prepared by including several important topics in computer literacy courses. First, the systems development life cycle concept should be included in these courses, with particular emphasis on the maintenance phase and its associated costs. Second, the role of perfective versus corrective maintenance should be explained, and the root causes of perfective maintenance explored. Third, prototyping as a method of addressing the communication gap between developer and client should be introduced.

The major implications of this research relate to the educator's responsibility to enhance the developer's understanding, however. First, the prospective developer must be at least as cognizant about the system development life cycle as his/her clients are. The educator's role in this

area, outlined in the preceding paragraph, is not diminished when addressing future developers.

Second, and probably more crucial to the prospective developer, is the need to understand structured systems design, the well-known benefits of which (isolation of errors, limitation of the effect of changes, etc.) are particularly needed in the prototyping environment.

There exists an even more fundamental reason for structured system design. To the client, prototyping is intentionally presented as highly flexible and adaptive. The tools that support prototyping (in this dissertation, primarily the ancillary tools associated with database management systems) reinforce this perception on the part of clients. This flexibility would quickly devolve into chaos if the developer did not have an internal roadmap to guide development. Structured systems design provides this internal structure. Moreover, a structure is needed to concentrate the client's attention at any given point in development. Most commonly, this structure is based on the menues of a menu-driven system. Educators must emphasize that it is not coincidental that the menu structure conforms almost perfectly to the internal structure of the system's design.

6.3 Limitations

We submit that this dissertation makes important contributions to the body of knowledge regarding systems development in general and prototyping in particular. Nonetheless, various decisions made in the execution of the

research reported here place significant limitations on the generalizability of the findings. The purpose of this section is to identify and to discuss briefly these limitations.

Two major limitations are present as a result of the domain specified for this research. First, this research deals only with application software intended to support business operations and decision-making. This class of decision-oriented software is generally termed "MIS" software. While prototyping has been extensively utilized in other fields, it is relatively new in software development. It is unknown to what degree the findings reported here are applicable to software development outside of the MIS application software domain.

Second, the research population for this dissertation is defined to be independent software developers who primarily focus on (1) MIS application software development in the (2) mini- and micro-computer environments, and who primarily utilize (3) database management systems and their ancillary products as prototyping tools. It is unknown to what degree one can apply the findings reported here to other development environments.

Other, less significant, limitations to generalizing the findings reported in this dissertation may apply. The research sample was drawn exclusively from the northeastern United States, and primarily from the prosperous suburbs of greater Boston. This region is marked by a "high tech"

economy. It is unknown to what degree the technological advancement of the region's economy has filtered down to the practical MIS software generated by the research population. There remains a possibility, however, that the research sample was somewhat more sophisticated than the general run of independent software developers nationally.

Further, prototyping is an intensely interpersonal experience. It is well-known that models of interpersonal interaction depend to a large degree on the culture in which the interaction takes place. Thus, it is unknown to what degree the findings reported in this dissertation can be applied to other cultures, both in other regions of the United States or internationally.

Despite these limitations, the present author contends that no great obstacles to generalizability are present in the research reported here, given the specified domain of the research effort. Therefore, one can envision that the findings can serve as useful information in addressing the prototyping phenomenon, and can serve to generate future research hypotheses.

6.4 Future research

The present dissertation is intended to enlighten our understanding of the prototyping process by addressing an aspect of prototyping, that of the developers' perceptions, that has been minimally addressed in the past. It is not intended to be definitive, and will properly serve its role if it generates ideas for future research hypotheses. We will address these future research directions in two ways:

first, we will highlight ways in which the questions addressed by this research should be expanded and/or replicated; and second, we will discuss other, related research that was not addressed in the present dissertation.

6.4.1 Research generated by this dissertation

In this dissertation we addressed for the first time the issue of delivery time of the system. We found that developers perceive that prototyped systems are not delivered in significantly less time than systems developed by other means. This definition of time, unlike a definition focusing on programmer-hours of development effort, has not been addressed previously, to the author's knowledge. Following scientific principle, others should attempt to replicate this finding. As an immature discipline, MIS has seen very little replication effort of reported research findings (Mahmood's work is thus especially important [Mahmood, 1987]). As the discipline matures, one hopes such replication efforts will become more common.

Our findings regarding the cost dimension, in which we found that prototyped systems are not delivered at significantly less cost than conventionally developed systems, contradict reasonable implications derived from previous research [see especially Boehm, et al., 1984]. Three possibilities are possible: (1) this research is correct and previous research is wrong; (2) vice versa; or (3) the various research efforts are not measuring costs in the same way. It is the feeling of the present author that the third

possibility is the correct one, specifically because previous research infers cost to be determined largely by programmer-hours of development, whereas the present research, by concentrating on developers' perceptions of cost, deals with a larger definition of cost, one that specifically includes long-term maintenance cost issues. The issue is further clouded by the developers' intuitive perception of maintenance cost, rather than any detailed accounting. Future research should address the cost issue in much more detail, possibly in conjunction with Managerial Accounting research.

Finally, the major finding of this dissertation, that a quality dimension motivates developers to choose prototyping, should be subjected to replication efforts. Further, our discussion arising from this finding raised numerous possibilities for future research. Specifically, the interaction of system quality and the financial operations of the consultant's business, whether by repeat business, word-of-mouth referrals for future business or by gaining long-term cost- and time-advantages, presents a rich lode of future research hypotheses.

6.4.2 Other research issues

Beyond the specific research interests addressed by this dissertation lie numerous other interesting research questions related to prototyping. This section will present two of particular interest to the present researcher.

Does the prototyping process, and specifically the interaction of developer and client, follow a known inter-

personal model? It would appear that several would be applicable, most particularly the Kolb-Frohman consulting model [Kolb and Frohman, 1970]. Research would be needed to: (1) operationalize the latent variables by establishing reliable manifest variables for each; (2) design an experiment to measure these manifest variables; (3) use a statistical tool capable of assessing latent structures to determine the validity of the model; and, should the model prove acceptable, (4) interpret the results. If it could be established that the prototyping process follows a known model of interaction, then predicting and managing the process would be surer, without taking away from prototyping the spontaneity that is its essence.

Further, does the frequency of changes engendered by the prototyping process follow a known probability distribution? Swanson's work [Swanson, 1988], drawing heavily from Brooks [Brooks, 1975] implies that the frequency of "bugs" in a new piece of software can be plotted. Accepting Lientz and Swanson's claim that a large portion of "bugs" are actually adaptations to users' needs [Lientz and Swanson, 1980A; see also the discussion in Swanson, 1988], which they term "perfective maintenance", then it follows that the continuous process of adaptation that is prototyping may also follow a distribution. If this distribution could be established, great progress would be made.

If both of these research objectives could be achieved, that is, it is determined that prototyping follows a known

model of interpersonal interaction and that the changes it engenders are predictable by the properties of a known probability distribution, then developers who prototype would have the necessary tools to bid on fixed-price contracts. Recall that the (at present) perceived unstructured nature of prototyping inhibits, if not prohibits, developers who prototype from bidding on fixed price contracts, and thus greatly restricts their business.

Prototyping, we submit, offers great promise as a reliable means of facilitating user-involvement in systems design and construction. Among the well-known benefits of user-involvement is the greater appreciation and use of the system by the people for whom the system was intended. It is hoped that the findings of this research effort will generate increased utilization of prototyping as an effective and efficient means of systems development.

EXHIBIT A

Interview Instrument

General outline

The interview consists of five parts:

1. Introduction
   a. Introduce self as interviewer
   b. Purpose of research
   c. Purpose of interview
   d. Define prototyping

2. Background data

3. Questions about prototyping generally

4. Questions about a specific prototyping project

5. Questions arising from the content of the dissertation

Transcript of introduction:

Good morning [afternoon]. I am Dave Russell, and I am a candidate for the Ph.D. degree at the University of Massachusetts at Amherst, in the School of Management. I am conducting research on the prototyping method of systems development. This research is part of my doctoral dissertation. As you may be aware, there are a variety of meanings to the term "prototyping", and one of the reasons I am interviewing you today is to ask your help in more clearly defining the term. I will give you my temporary, working definition of the term in a moment.

First, I want to thank you for inviting me to your office [home] today so that I can include your views and experiences in my research study. To gather these, I will interview you in a structured way, that is, I will ask you a specific set of questions in a specific order. I am interviewing you in this way in order to have a common base of comparison between your views and the thirty or so other people I will interview for this study.

This study is seeking the answers to two major questions. A major question to be answered is this: does the prototyping method of systems development deliver systems in less time or at less cost than does traditional development? Of equal importance is this question: do developers like yourself perceive that prototyped systems are of higher quality, about equal quality, or lower quality [reverse every other interview] than systems developed traditionally?

This interview has been structured in a way that will attempt to obtain clear answers from you and the other interview subjects. Your views and the views of the other interview subjects will be the basis for much of the findings of this research.

We will proceed in this manner. First, I will give you our working definition of prototyping. You have been selected for an interview because you have been identified as an active prototyper. But, your definition may well differ from mine, and I want to seek your definition in a subsequent question.

Next, I will ask you several questions regarding your professional background and experience. These questions are necessary to help us analyze differences among the various responses we will receive to this interview.

Next, I will ask you several questions regarding your general views about prototyping, beginning with your definition of prototyping.

Finally, I will ask you a series of questions regarding a specific prototyping experience you have had.

To conclude, I will ask you one or two questions regarding how you would like your participation in this research credited to you.

This interview will take approximately 45 minutes more. Is there any way we can minimize interruptions?

May we begin now?

INTERVIEW BEGINS

A moment ago, I promised you that I would share with you our working definition of prototyping. As one of the purposes of this research is to more clearly and explicitly define prototyping, you won't be surprised to hear that our definition is very general.

We define prototyping as follows: prototyping is a conscious attempt to deliver a running version or parts thereof to a client very early in the development process. In doing so, it is recognized that the "rough draft" could be wrong: what is sought is the client's reaction to the draft. The client's reaction to the rough draft then guides the next version of the system. The process repeats itself until the client is satisfied.

We intentionally want to contrast prototyping with the traditional development method. In the traditional method, much effort is expended in analyzing the needs of the user

and subsequently designing the system before the system itself is created. Thus, the client does not see the system itself until much later in the development process.

I realize that you may use a different term for this development technique. Could we agree to use the term "prototyping" for purposes of this interview?

I would like to use a dictating tape recorder to capture our conversation. This will allow me to concentrate more fully on your comments. Will that be all right?

BACKGROUND QUESTIONS

1. First, let me show you your name, company name, address and the like as I have it on file [show copy of acknowledgment letter]. Is this information correct? Have I spelled everything correctly?

[Exchange business cards]

2. Let me ask you a few questions about your professional background.

2a. What is your educational background, please?

High school?

College?
Graduated?
Major?

Graduate school?
Graduated?
Major?

2b. Judging by the information you just gave me, I would say you are in your (early 20's, late 20's, early 30's...). Is that correct?

2c. When did you begin your professional career in MIS?

2d. Could you give me a brief summary of your MIS experience to date?

2e. From the information you just gave me, it seems that you have had approximately __ years of professional experience. Is that correct?

2f. Could we break down your professional experience a little more exactly?

2f1. How many years in primarly mainframe computer environments?

2f2. How many years in primarily minicomputer environments?

2f3. How many years in primarily microcomputer environments?

2f4. How many years of non-administrative experience?

2f5. How many years of administrative experience?

3. Would you now share with me your own operating definition of prototyping?

4. When did you become aware of prototyping as a conscious method of systems development?

5. When did you begin prototyping consciously?

6. I will now ask you some questions regarding the goals you have for your MIS career. Then, I will relate these questions back to prototyping. What are your short-term goals in your MIS career?

7. What role does prototyping play in your short-term MIS career?

8. What are your long-term goals in your MIS career?

9. What role does prototyping play in your long-term MIS career?

10. How many systems have you prototyped to date?

10a. What percentage of the systems with which you are actively involved are prototyped? Please answer in increments of 10, that is, 10%, 20% ... 100%.

10b. What percentage of all the systems with which you are indirectly involved are prototyped? Please answer in increments of 10, that is, 10%, 20% ... 100%.

GENERAL ATTITUDES RE: PROTOTYPING

1. I will now make several statements. Please tell me if you Strongly Agree, Agree, Disagree, Strongly Disagree, or are undecided about each. [Note: reverse agree/disagree order every other interview]

1a. User-involvement in systems design results in increased system usage.

1b. User-involvement in systems design results in increased user understanding of the system.

1c. User-involvement in systems design results in increased appreciation of the MIS function in general.

2. Assessing all your systems development experience, industry-wide, to what degree would you say users had substantial involvement (answer in percentages, in increments of 10%).

3. Are you more likely to prototype now than:

3a. [if relevant] 2 years ago?

3b. [if relevant] 5 years ago?

3c. [if relevant] 10 years ago?

3d. Why?

4. Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that are successfully prototyped?

5. Please think of systems that were not successfully prototyped. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

6. Let's discuss the time it takes to develop system. Here, I refer to the calender time from conception to delivery of a system, not manhours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner?

6a. Why?

6b. On what do you base your opinion? Personal experience, internal studies, consultant's advice?

6c. Any factual evidence?

7. In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner?

7a. Why?

7b. On what do you base your opinion? Personal experience, internal studies, consultant's advice?

7c. Any factual evidence?

8. In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality [reverse every other interview] that those develped using more traditional approaches?

8a. Why?

8b. On what do you base your opinion? Personal experience, internal studies, consultant's advice?

8c. Any factual evidence?

9. Some people feel prototyping is an effective method of understanding user needs. Others feel that more traditional analysis and design methods are more effective way of understanding user needs? [Reverse order every other interview]. What do you think?

10. Some people feel that prototype should consist of simulation of a proposed system, with no actual underlying computations coded. Others feel that actual computations should be included in the prototype? [Reverse order every other interview]. What do you think?

11. Limiting our attention to prototypes that involve actual computation, some people feel that a prototype should be discarded after it serves its purpose. Others feel the prototype itself can evolve into the production system. [Reverse every other interview]. What do you think?

[11a. If answer indicates evolutionary development: some critics argue that an evolved system is executionally inefficient, and that recoding is necessary for performance reasons. What do you think?]

12. What tools do you use for prototyping?

[Be sure to cover:]

12a. Simulation tools?

12b. DBMS?

12c. Piecemeal assembly of components?

12d. Prototyping tools, e.g., Clarion?

13. Where should prototyping tools be placed?

13a. In the operating system?

13b. As separate application development environment?

14. How focused are you on a single prototyping tool?

14a. Any concerns re: enhancements, support of that tool?

14b. Will your prototyping tool become obsolete?

14c. How do you judge the stability and longevity of the firm from which you purchased your prototyping tool?

14d. How much do you have invested in your prototyping tool (by this I primarily am concerned with effort to learn the tool well)?

14e. [If 14d indicated substantial investment]. What would it take to cause you to leave your prototyping tool?

15. There is some indication that OS/2 will include a DBMS, a screen painter, and other prototyping tools. Should these be acceptable tools, what is your reaction to this case?

16. What features do you desire in a prototyping tool that are not currently available?

17. What shortcomings do you see in the prototyping method?

18. The next question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping the the operation of your business, particularly the financial operations?

19. Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

SPECIFIC QUESTIONS RE: PROTOTYPING

1. Are you currently involved in a prototyping project?

2. For this project, or the last project prototyped if you are not currently involved in a prototyping project, what is its:

2a. Would you consider this to be a typical project? [If response is "no", request that respondent address a typical project]

2b. nature?

[In the next three questions, I will ask you to scale your answer from 0 to 7, as I will indicate on each question.]

2c. size, as measured by: 0 = very small to 7 = very large?

2c1. To scale your answer, what size would represent a "3"?

2d. complexity, as measured by: 0 = very simple to 7 = very complex?

2e. cost, as measured by: 0 = very inexpensive to 7 = very expensive?

2e1: To scale your answer, what cost would represent a "3"?

3. Why did you choose to prototype this project?

4: Imagine this project had been performed as a traditional development.

4a. How much less/more time [reverse every other interview] did the prototype take?

4b. How much less/more cost [reverse every other interview] did the prototype require?

5. On this project, what aspect of your prototyping system did you work on? Screens, Report generation, Database, or what?

6. Based on this particular experience, what was the single greatest strength of your prototyping tool?

7. Based on this particular experience, what was the single greatest weakness of your prototyping tool?

ADAPTABILITY TO PROTOTYPING

1. When you are involved in prototyping, how do you approach the project? What is your plan of attack?

2. All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious [reverse every other interview] than traditional development?

3. When prototyping, do you find you must take a leadership role with respect to clients more often or less often [reverse every other interview] compared to traditional development?

4. In general, when prototyping, do users want to you take the lead in presenting options or do they bring up sufficient options on their own?

GENERAL QUESTIONS

1. When this research is completed, it will be presented in a doctoral dissertation that will be available to the public. It is also possible that this research will be presented in professional journals. How would you like your participation in this research credited?

1a. Not at all: keep my participation confidential

1b. List my name in an appropriate place

2. Should I quote some of your points, would you like to be credited by name?

3. If not, may I use your initials to identify you without giving your name, for example, Mr. (Ms.) __ __ noted that...?

4. I would now like to continue running the tape recorder and gather other thoughts you have.

[Thank respondent and promise to send summary report in August]

END OF INTERVIEW

EXHIBIT B

Interview Evaluation Instrument

Respondent:

Date:

Time:

2a High School? ___yes ___no

College? ___yes ___no

Graduated? ___yes ___no

College:_____

Major:_____

Year:_____

Grad School? ___yes ___no

Graduated? ___yes ___no

College:_____

Major:_____

Year:_____

2b Age:_____

Early 20's___ Early 40's___

Late 20's___ Late 40's___

Early 30's___ Early 50's___

Late 30's___ Late 50's___

2c Year began MIS career:_____

2d Experience represented:

Programming:_____

Analysis:_____

Project management:_____

Other management roles:_____

Training:_____

Other technical roles:_____

2e Years of professional MIS experience:_____

2f Mainframe:_____

Minicomputers:_____

Microcomputers:_____

Non-administrative:_____

Administrative:_____

3 Definition of prototyping [response transcribed]

4 Aware of prototyping: ____ (years)

5 Use of prototyping: ___ (years)

6-7 Does prototyping play a significant role re: short-term?

___yes ___no

8-9 Does prototyping play a significant role re: long-term?

___yes ___no

10 Number of systems prototyped to date (est.):_____

10a Percentage development experience using prototyping ___%

10b Percentage non-development experience using prototyping ___%

Other background (not specifically questioned):

___ PT developer/has FT job elsewhere
___ FT entrepreneur
___ FT staff member/small firm
___ FT staff member/large firm

GENERAL ATTITUDES

1a ___SA ___A ___D ___SD ___U

1b ___SA ___A ___D ___SD ___U

1c ___SA ___A ___D ___SD ___U

2 Industry-wide perception of user-involvement: ___%

3a More likely than 2 years ago? ___yes ___no ___irrel.

3b More likely than 5 years ago? ___yes ___no ___irrel.

3c More likely than 10 years ago? ___yes ___no ___irrel.

3d Why? ___better tools

___better skills

___combo of better tools & skills

___other

4 [Transcribe response]

5 [Transcribe response]

6 ___less time ___more time ___same

6a [Transcribe response]

6b ___personal experience

___internal study

___consultant's advice

___other

6c ___yes ___no

7 ___less cost ___more cost ___same

7a [Transcribe response]

___personal experience

___internal study

___consultant's advice

___other

7c ___yes ___no

8 ___lesser quality ___more quality ___equal quality

8a [Transcribe response]

8b ___personal experience

___internal study

___consultant's advice

___other

8c ___yes ___no

9 ___Proto. more effective

___Traditional more effective

___Depends

[Transcribe response]

10 ___Simulation

___Coding included

___Depends

[Transcribe response]

11 ___Modelling

___Evolutionary

___Depends

11a Recoding? ___yes ___no ___depends

[Transcribe response]

12 List tools:

12a Simulation tools? ___yes ___no

___depends ___used for other than proto.

12b DBMS? ___yes ___no

___depends ___used for other than proto.

12c Piecemeal assembly? ___yes ___no

___depends ___used for other than proto.

12d Proto. tools? ___yes ___no

___depends ___used for other than proto.

13a Proto. tools in OS? ___yes ___no

___depends

13b Proto. tools in separate applications development?
___yes ___no ___depends

14 Focused on a single proto. tool? ___yes ___no

14a Concerns re: enhancements, support?

___yes ___no

14b Becoming obsolete? ___yes ___no

14c [Transcribe response]

14d Investment: ___minimal ___moderate
___substantial

14e [Transcribe response]

15 Reaction to proto. tools incorporated in OS/2:
___enthusiastic
___receptive
___cautious
___negative

16 [Transcribe response]

17 [Transcribe response]

18 Impact of proto on financial operations:

___minimal ___moderate ___substantial

[Transcribe response]

19 Can experience substitute for proto.

___yes ___possibly, limited ___no

[Transcribe response]

QUESTIONS RE: SPECIFIC PROTOTYPING PROJECT

1 Currently involved? ___yes ___no

2 For a chosen project:

2a Typical? ___yes ___no

2b MIS application? ___yes ___no

2c Size perception:

3 = _____ (MM)

2d Complexity perception:

2e Cost perception:

3 = $_____

3 [Transcribe response]

4a Proto. vs. Trad. re: time:

___less ___more ___same ratio:

4b Proto. vs. Trad. re: cost:

___less ___more ___same ratio:

5 Aspect of proto. tools most used:

___screen generation

___report generation

___DML

___other:_____

6 Single greatest strength:_____

7 Single greatest weakness:_____

ADAPTABILITY:

1 List steps sequentially:

1.

2.

3.

4.

5.

6.

7.

8.

9.

10.

11.

12.

[Transcribe response]

2 ___less anxious ___more anxious

___not answered

[Transcribe response]

3 ___ leader more ___leader less

___ not answered ___about equal

[Transcribe response]

4 ___must present options ___user brings up enough

___ not answered ___about equal

[Transcribe response]

Conclusion: [Transcribe response]

EXHIBIT C

Transcript of Responses to Selected Interview Questions

Respondent 1

Would you now share with me your own operating definition of prototyping?

Prototyping to me would be showing... 90% of the software in this business is input and results. The easiest way or the best way to do prototyping is the design of the screen, showing the placement of the fields, where the information would go in, somehow simulating that through [the] use of tools that are available to us, where we do replays of things we do on the screen. Then showing them the results, which would be the reports or the orders or things like that. And, it's just done with dummy information, basically. So, there is just designing the screen, showing the user interface, showing how it will look for the user interface. Now, as far as the actual content, it's done through means of a flowchart type situation, where you would sit down with the customer and decide what information has to go into the system -- just showing the route where it would go, generally it's a main menu... with submenus off that. You can just show them those menues and if nothing is coming off that menu, it's not really a big point. So you can generally show the menus and the main input screen. And from that point, you show them the results, which would be the reports.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

No.

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

User didn't know what they wanted and what was provided just tended to confuse them even more because of their non-computers situation. People just sit down with a blank screen and get totally confused, no matter what you do.

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

I don't really think it makes a difference. Because whether or not a customer wants or gets something prototyped, it doesn't usually change the outcome to the point to where... if the customer wants something prototyped, and you give it to them prototyped, it takes X amount of programming to produce that. If they don't want it prototyped, then it's up to me to give them something, and generally it takes about the same amount of time, because programming is programming. You still have to end up with the same results, whether I show them in the beginning what it's going to look like, or whether I just go ahead and do it... I guess mainly prototyping will help the customer understand what it is they are going to get. I don't see any advantage in time, calendar time... in fact it may take even longer because the prototype will take a certain amount of time to do. I have to distinguish the type of customers. Customer who require prototypes tend to take longer to decide things, they do more research, they discuss things more and from the time I show them something to the time they even decide to do it could be three to six months, as a typical thing. Whereas [if] a customer says 'I don't care, I trust you. Go ahead and do it.' At that point you can start the project that day and deliver something in six months rather than talking about it for six months.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

[No specific reply.]

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

I would say the quality is greater, because... programming is like being an artist. If you prototype something, that means you have set certain guidelines that you have to meet. The customer expects that and you have to meet those guidelines. If you do it on the fly, you are basically free to do whatever you want to do. You tend to cut corners sometimes.

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

I try to simulate as much as I can what they [customers] are doing manually, right from trying to simulate their own worksheet on the screen, the flow of the order process to go on with what they perceive as a good flow. At the same time, being a consultant in the order flow, to the point where if I see a problem in their order flow, we'll try to use the software to correct that, to make it more streamlined. But that's the way I get at their needs, is to go into their company and see exactly what they're doing now and try not to change things too radically because you run into a lot of ego problems too.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

[It] makes no sense to include the computations, because that's the easy part. The computations actually tend to be the easy part, that's all in the program underlying. The hard part is getting the user to like to screen, to follow the cursor in a way that they can easily input the data. I find that that is 90% of the battle: getting a comfortable flow to the input screens. And you can do that with the simulations, that's what you need.

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

Well, if you're prototyping in a way to try to sell something to somebody, that to me is putting in a lot of time with the possibility that it may not even be what is it looking for. If you can give them a simulation... Actually, the simulation turns out to be more useful in the program than actually doing the computations. Again, the hardest part is designing the screen. Computations are easy. The screens can be used because they are one little packet of code [implies saving screens and transferring to final program].

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

It's not as major as you'd think. The tools are very reasonably priced because they are selling to a mass market. And, there's always new ones coming out, better than the

previous ones. And the learning curve is generally not prohibitive. So, you're not making a major investment of time and money. So, you tend to... you see something you like and you buy it and you add it to your toolbox. And if you don't use it you say 'So what, I spent $149 and it's part of the game'. You do it that way and you'll eventually find three or four good ones that you can use.

What would it take to cause you to leave your prototyping tool and move on to another?

Well, if I was strongly convinced that that language was headed for obsolescence because something else was better. And "better" has a two-folded meaning: better as a physical product or better means more people use it, which makes it better. If both of those things were true, I would consider moving to that thing that was supposedly better. I've done it with word processors...

What features would you desire in a prototyping tool that are not currently available?

They're still not easy to use. You've still got to a lot of... But I think they pretty well offer everything you'd need to convince a customer or to show a customer what it is they [the customers] need. They really don't need to see that much... I haven't gotten up to the bit, to the top level yet, I'm [consulting for] small to medium businesses, up to $10 million-a-year type business. So, I really haven't experienced the ultimate in going in and designing a system and going through all that, so, I may not be able to answer that as well as I should. But, for my purposes, I have available to me anything I want to buy. I may not have it myself, but I know it is available, and I just don't have it because I don't see that much of a need for it.

What shortcomings do you see in the prototyping method?

In the methods that are available, I don't really see any shortcomings. I think that they [prototyping methods] are providing what's needed, although I've never really seen anything that specifically says 'This is a prototyping tool'...That would be kind of nice. I've yet to find a prototyping tool that would take a screen, write it to a disk file, so that you can include it in manual...

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

[Not transcribed.]

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

If you have no experience with the product or the system you are providing, prototyping would help me as a developer understand what it is they want... If I had no experience..., prototyping would help me understand because it would force me to sit down with the user, figure out what it is they need. However, if I've already done it, then I can sit down tell them what it is they need and they may not need a prototype based on the fact that I have knowledge of what it is they need, or I have done it before, or whatever. So, in that case, prototyping may not be as needed.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

Because the customer had to... they had an idea of what they wanted. And, this was one of my newer customers, who kind of hired me out of nowhere and that I had to show them that I knew what it was that they wanted, or that I understood what they wanted, before I was willing to make a commitment of time, of programming, I had to make sure that they accepted my ideas. I was in there as a consultant also, so I had to change things around a little bit.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?.

[Not transcribed.]

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

[Not transcribed.]

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

For the most part, I take the lead. I think that's only true, though in the initial to 50-70% complete stage. At that point, they now have a better feel of what the system can do and they start presenting options to me of things they'd like to see.

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

Just that... my opinion of where prototyping is going is that... it does not seem to be the factor that drives companies to have software written for them. The thing that drives them is the assurance by somebody like me who they have either heard is good or they've had experience with, the assurance that the computer will mimic or automate what they are doing manually. If you can assure them that they won't skip a beat and they'll just be able to continue to operate with no major changes and no headaches, then that's what they're really looking for... They just want to know that everything is going to go smooth[ly] and instead of having to do everything manually now they're going to cut hours by computerizing. To me, that's what's really becoming important in this business... Once you get the base of the system in there, they're confident with it... That's why canned packages are not making out very well these days -- it's the custom dBase stuff... That may be replacing the prototyping, the fact that I can show any one of the [existing] modules... and say that this is the way it is right now, now this can be changed.

EXHIBIT D

Transcript of Responses to Selected Interview Questions

Respondent 2

Would you now share with me your own operating definition of prototyping?

It's probably similar to your working definition, where after an initial jab, which is what [the respondent's organization] coins a "Joint Application Design Effort" ("JAD"), between the user and the Data Processing support staff, we come up with a paper prototype of what functionally a system should do. At that time, the developers go back and put together the bare bones skeleton, operating environment on the computer, which is pretty much what you said. It's a mockup. With some assumptions in mind we put together what we perceive is a functional prototype system - what we perceive that to be. And the only way you can elicit comments from the user community is if you present something for them to comment on. So, I guess the functional definition of a prototype is that: it's a working product of what was discussed in the JAD session. The sole purpose of eliciting ideas to refine the system.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

I would say that the main thing that sticks out among system with active user involvement: the systems have inherent ease-of-use feature and a lot of on-screen documentation. These are the two things that users have a great impact on when it comes to actually prototyping.

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

Probably they don't do what the user wants them to do. Not only that, they probably don't function the way the user wants them to function. It's usually a situation where, depending on the people involved, from Data Processing -- you've got to understand that there are a lot of people in Data Processing who follow the old rules where DP [Data Processing] comes down and basically blesses the users with a system. The DP [Data Processing] people, a lot of times, tend to get frustrated with user-involvement for one reason or another because they don't understand the guts of the system and how it's supposed to work. In those particular instances where the DP [Data Processing] people take a lot for granted and deliver systems without a lot of user-in-

volvement you find users a lot less willing to use the system -- it's just the way things are.

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

What you find when you go back and forth with different beta version of a product for the user to test out is that, although your end-result is a system that works functionally like the users want it, you are constantly refining the system to meet the needs of the user. Those refinements take time. What tends to happen, especially in a reporting function where you have output that users want to see, you go 'round and around and around and around many, many, many times before you come up with what they they like or that they're going to be able to use. That takes a lot of time, as opposed to just having Systems just deliver a product and drop it off at the user's doorstep.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

My reaction from a near-term cost perspective is that they [prototyped systems] are more expensive because development time goes up. From a long-term maintenance perspective, they probably will work out to be less expensive because there will be less changes that the system has to undergo after initial delivery.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

[Not asked.]

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

I would have to say that today you're getting a mix of those traditional development efforts along with heavy user-involvement. I still think in a lot of instances, especially in large systems development, traditional flowcharting, systems design and things like that are done, but there's also a heavy amount of user-involvement. I don't think it's one or the other.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

[Did not answer.]

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

[Did not answer.]

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

[Did not answer.]

What would it take to cause you to leave your prototyping tool and move on to another?

[Did not answer.]

What features would you desire in a prototyping tool that are not currently available?

Some of the things that are now becoming available, especially with [80]286 and [80]386 [computer chip] technology is the increased use of a graphical interface to computing: the point-and-click techniques that are used with the Macintosh and traditionally developed by the people at Xerox at the Palo Alto Research Center were adopted by Apple initially in the development of the Lisa and the Macintosh and picked up on by MicroSoft and brought over to the IBM world now under MicroSoft Windows. IBM and MicroSoft jointly write operating systems -- MicroSoft generally writes the operating system and IBM helps co-market those things and develop the machines. What you're seeing now is the very beginning towards the development of a graphical operating system: Presentation Manager under OS/2. And, a lot of graphical based development and user-based tools that will work in that environment. Prototyping tools, just like development tools, and user-tools are going to fall along those lines, where maybe instead of writing code from scratch, we will see (and I've got samples of some of this stuff) the use of the mouse and the graphical interface to build programs and actually develop custom menues, pull-down menues, dialog boxes for developers to use in their prototyping efforts. That's the direction.

What shortcomings do you see in the prototyping method?

I would say it could be in how different people use prototyping to get to the final product. What often happens is: people will develop the prototype, get feedback from the user, and go ahead and develop the final system. But what really needs to happen is a continual exchange of ideas, devise prototype systems for the users. Some people just say: I've developed the prototype and the users liked it, so now I can develop my final system.

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

[Question inappropriate.]

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

I don't think there is any substitute for prototyping, to be honest with you. The experience of the programmer in a specific development environment, and also with other users who have helped him assist in prototyping in that same environment, might help to move the development process along a bit. He's been through it with that prototyping environment. But, each user and each system is different, and although the methods may remain the same and the products may remain the same, the prototyping effort should pretty much depend how how the users react to the system and how complex the system is.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

There is no way that you can deliver a system, in my opinion, to a user, without having them actively evaluating a prototype system. They have to use the product, to get the feel for the product, before they sign off.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

[Not transcribed.]

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

[Not transcribed.]

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

[Not transcribed.]

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

I would just caution you that it's hard to judge the success of prototyping by looking at development time or development cost or by looking at developer's perspectives. ...Don't look at short-term cost-benefits. Try and look at it from a longer-term perspective because it's my opinion that the whole prototyping process is the involvement of the users is a pain in the neck for the developer, and it's probably a pain in the neck for the user. It results in a little bit longer lead-time and a little bit more expense. But typically what you get out of it is a better system, one that requires less ongoing maintenance. So, the ongoing costs, the ongoing involvement is less. The initial up-front costs are a little bit higher. ...A lot of people think, traditionally, that Systems develops a product and delivers it to the user, and let the users use it for a while, and then at the next revision, what they consider using a production system as an ongoing prototype. I think the value of prototyping is that you try the best you can to work the bugs out in the prototype version so that when you finally do have a deliverable system you've got one that has inherent flexibility, one that needs less ongoing maintenance. And, it's less of an ongoing support issue that it would be if it were done in a traditional way. That's my basic opinion on it. what we're seeing happening is really a movement from this technological revolution from more traditional development environments to smaller distributed development environments. The technology that's coming out now, we're seeing it all over the place, has allowed for development to be done at the user level. We've got DP [Data Processing] people going out and actually working in user areas. They're doing development that is specific to user needs because they work with them all the time as opposed to being in a centralized DP [Data Processing] shop. They're going to be using [80]286, [80]386 and [80]486 [computer chip]-based technologies inherently two or three times the power of any minicomputer you see out there now. The graphics tools, the program development tools and the general complexity of the systems are going to be, we're going to be able to get much more sophisticated with them because of the technological capabilities. So, I think what you're going

to start seeing is a much more decentralized approach to development, as opposed to the centralized. It's happening now, because the development environment has been more conducive. I think that's a trend, and that's a good trend, because in a lot of cases developers are users themselves. You're seeing users who have taken to microcomputing to such an extent that they are being experts themselves. They're doing the development; they've got an inherent working knowledge of the business function and an inherent working knowledge of the technological capabilities. You can start to see development from non-programmers; non-programmers development tools are starting to come out.

EXHIBIT E

Transcript of Responses to Selected Interview Questions

Respondent 3

Would you now share with me your own operating definition of prototyping?

I guess as I listen to yours [the interviewer] my definition is largely the same: to develop a subsetted but working system early in the process. The only thing I might add to your definition is that often we find that we do that as a proof of feasibility in addition to proceeding with the assumption that the system can in fact be built.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

Probably the most important characteristic among the successes is that the need for the system was valid in that it was initially fairly well defined.

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

The other side of that [refer to response to success question]. If the user doesn't have a clue as to what he wants, prototype will not help you.

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

Makes the system-building process concrete. The analysis and discussion phase can waste a lot of time arguing abstractions.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

[Response is that prototyping probably allows systems to be developed at less cost than non-prototyped systems.] Well the qualification there is based on size and scope. The things that we do here, because we're real big, are systems that are real big, and so the amount of resource that you tie up in analysis phase and the staff time is significant.

If it were a small PC [personal computer] application built by one guy to serve one guy, then the prototype actually might be more costly.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

Generally greater quality. One qualifier on that: it depends on how you move from the prototype to the final system, whether you do it by a rewrite or whether you do it by a head-on development.

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

[Not transcribed.]

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

It's a tactical issue for an MIS Director. How I would do it would depend on how confident I was at getting money in the time necessary to do essentially the rewrite, as opposed to being forced into running the prototype as a production system and adding onto it.

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

I think you should throw it away... Because part of what you're trying to do when you create a prototype is to explore alternatives, to get something up and running quickly. Those strategies cause you to make choices that aren't necessarily the right ones for... systems whose anticipated lifespan is long. Plus, you sacrifice part of the learning: part of building the prototype you will learn a better way to build the system. If you don't then rebuild it, you're sacrificing what you learned.

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

Standard process: look them up, talk to other users, look up reports in the trade press.

What would it take to cause you to leave your prototyping tool and move on to another?

[To leave his prototyping environment he would need] some actual experience that the new one is more effective than the old one.

What features would you desire in a prototyping tool that are not currently available?

I guess a graphical representation of the code and the data objects would help a lot... Certainly the AI that they're trying to build into some of the tools has some promise, but it's two or three years away.

What shortcomings do you see in the prototyping method?

A couple of shortcomings, I guess. One is, when you do the prototype, you run the risk that you'll wind up putting your prototype into production, and thereby have a worse system than you might otherwise have. The second is, the users will become too quickly invested in the system, and miss opportunities for substantially better system that they might have discovered through a longer process.

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

Sometimes it does, sometimes you need to produce a prototype as a proof of feasibility before you can get the major project funded.

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

I guess I don't know. I look at that issue substantially differently, and that is to say if the developer has substantial experience then I might be more likely to prototype, because it is more likely that I can get something which is substantially correct done faster.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

Proof of feasibility as much as anything else. Plus politics.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

It's fairly situational, I think.

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

[Not transcribed.]

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

[Needs to take on leadership more often] because of everybody's desire to put the prototype into production.

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

[No specific reply.]

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

One of the things that we need to do, I think, is to put most of my comments in a context. In order to do that, you've got to understand that [respondent's organization] is very large - it's Fortune 11. Most of the experience I'm drawing on are the last four years which are here at [respondent's organization], including systems which are significantly bigger than most other people have experienced... Scope definitely comes into play. You can't effectively develop a prototype for systems in an environment which makes it available to [many thousands] of users. There's a fundamental conflict right there. If we do our prototype in FOCUS, which is highly likely, there's no way we can put it on for all [many thousands] users. It just consumes too many machine resources... So we're in an environment that is somewhat structured to prevent us from rolling over prototypes into a working system. The other part of the issue that intervenes is: how structured your development organization is. Our organization, in the tradition [of the respondent's organization], has very little structure... If you're trying to do a prototype in an unstructured environment, where mobility is high, you tend to push for minimum development time, as opposed to strong initial design even though you are building a prototype. So it's important to us, given that we've essentially optimized the time-line to then go back and do a solid design of the internals; that again orients you toward throwing away your prototype. In the PC environment, particularly if you are doing something with a database package, then it's much less true because

the package is going to take care of doing all that housekeeping for you and structure your internals anyway... In general, I think if you've real good people on your team, that they will in fact learn a lot if given the opportunity to learn a system once, throw it away and start over. And the real ace developers that I've had the opportunity to work with generally don't lose any time by doing that: generally you will have the system memorized and can change it in their head and can essentially recreate it from memory... You constantly walk a fine line between automating what a user does now and going in and doing a full-fledged M&P [methods and procedures] study of what's going on and automating the right thing. One of the risks you run with prototyping since [you are] developing a system as early as you can, is that it orients you in the direction of automating what the currently workflow is, when in fact the appropriate solution is to redesign the workflow along with building the system. That's a constant thing you have to balance off and it is one of the drawbacks of prototyping... particularly if the person doing the initial analysis and design is inexperienced in the area where the user is working, then [there is] a strong tendency to just automate the status quo... I think there are clear trends in the way things are going... I think things are trending toward prototyping, things are also trending toward user developed systems. The trend toward prototyping among MIS professionals is likely to help systems development. The trend toward user developed systems and prototyping in that area is likely to hurt long before it helps. The users obviously don't have the professional experience that developers do, so they're going to spend a lot of time fumbling around, until you get in some really strong tools with a lot of AI [Artificial Intelligence] for user developed systems...

EXHIBIT F

Transcript of Responses to Selected Interview Questions

Respondent 4

Would you now share with me your own operating definition of prototyping?

Pretty close to what you describe, except that I think that our development method is probably more of a mixture of the old up-front, heavy planning and the prototyping [in that] we'll have a series of meetings with clients and just keep pushing until we feel we have a solid understanding of what they want. We will deliver simultaneously then a specifications document and a working prototype that will have menu structures and couple of sample entry screens so that they can get a flavor of what the entry screen look like. This specification document is usually very, very ...as precise as we can get it. Certain standard ways screens interact, sample layouts, actual layouts of reports, all that kind of stuff is included in the document. What we're trying to do is to minimize the kinds of changes in the modification process. I would say that we are not intentionally delivering imperfect stuff, no, that's not what we do. We deliver a document that contains our fullest understanding of what we want, in great detail. Along with that a working prototype that's working in only a few places. When you pick a menu item, it will say "And what you'll see eventually..." and just describes it. But there will be a couple of working entry screens so they can get a flavor of what it is like. I'd say that, in that sense, we're sort of a balance between the "give it to 'em quick 'n dirty" and "plan it thoroughly" approach. We are planning it thoroughly. The actual first serious delivery we make is intended to be working: if there are bugs in it, they are not known. They're not bugs that we know are there, or if we know they're there we'll tell them [clients]. It's not a deliberate attempt to throw out something with mistakes in it.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

There were lots of commonalities. Clients had an overview of what the process was going to be in the first place, so they knew what to expect in terms of time, in terms of what you were going to expect from them and what they were going to expect from us. The care with which we try to understand what they wanted led to... holding of meetings. In those meeting, we're playing a dual role of listening very carefully, but being very hardnosed about questions, pushing people to be specific when they were being vague. We made it very clear that we couldn't develop until we know what reports they wanted. It didn't make sense for us to do

anything until they could specify their reports. That often meant waiting several months until they came up with their reports. The next common factor was the development of a specifications document, which is an art we're still trying to polish, but the factors in there are a text description of what we understand they are all about, why are they doing this anyway, what they want the system for, an outline of a menu system and what they can expect will be an action for each menu item, some sample forms on paper. Most important, in great detail, report samples indicating what options they'll get in printing it, what the layout will be, sort order, as well as technical notes for people doing the actual development as to how this will be set up. We also attach to that a design, a database design, on paper and on disk, and we also will do an estimate of how much space the database when completed to take up so that we can do hardware recommendations along with it. The next step is to deliver that prototype that I mentioned. The next step is a meeting to go over the document, answer questions, go over the onscreen specs and get initial reactions to them, which are often very helpful. Usually we are pretty much on target at that point, but if there has been any slight misunderstanding we can correct it there. And then we work out a schedule of delivery. We also give them control over the project task by task; it's not a simple matter of sign a contract here and now send us off to work. We estimate how long each task will take and they have to sign off on each task before we'll do it. Once they've signed off we'll go ahead and deliver in stages. If at any point while we're working on the next stage they want some fixes on the first stage, we'll go ahead and do those, but my experience is that the fixes tend to be either one of two categories. Either very minor corrections based on a slight misunderstanding or they've changed their minds or major ones because they really didn't have enough experience with computer themselves to understand what they could do. Once we give them something and they work with it they begin to think of other things they'd like to have that they didn't ask for originally. And that sometimes requires major changes, but there is nothing you can do about that.

<u>Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?</u>

We didn't give them enough of a prototype early enough. I did a paper prototype; I didn't do a disk prototype. And there was one case were we did a lot of work on a project and we never really delivered a working version until they got one, and within two days they were clear that that wasn't what they wanted at all.

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

It's really based on the horror stories people tell you -- a year-and-a-half waiting list for MIS to get to you. You get a quicker sense of what they want and a quicker sense of when you're on the wrong track, so you don't waste a lot of time going down blind alleys.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

I'm not sure. I have not actively sought to compare our prices with other people; I don't know if we're charging more or less. I known people are satisfied with what we're doing, and I know my goal is to be able to do it for less, but I can't tell you factual figures.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

You get an earlier confirmation of whether you and the client are on the same wavelength. You know computers, one of the things that makes computers so attractive and so useful is the immediacy of feedback. That's why they're such a great learning tool... Prototyping is just a natural extension, taking advantage of the computer's ability to do that. You understand that it's very different to hear about something, and to talk about something until you see it and use it. So no planning process is complete until people have a chance to see it and touch it and view it.

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

I see it as a confirmation of understanding user needs -- the prototype itself doesn't... It has two functions. It has the function of confirming we've been on the right track but it also has the function of, early on, because people will tend to perceive things differently after they have used it, as opposed to just talking about it, then it can have the function of helping them recognize things they wanted that they didn't even know that they wanted and that they needed. But it doesn't in any way substitute for careful discussion and listening and questioning.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

In that first stage prototype, as I mentioned, we won't do much [computing]. There might be... It depends on the project: I'm really after more of giving them a quick flavor, of what it means to sit down and use a database through a menu system. The next step is delivery of a working piece of the product.

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

We're beginning to lean toward that [recoding for efficiency considerations]... As the community of users with R:base [respondent's prototyping tool] has become more sophisticated [in] the demands they are making of us have been more sophisticated. It began to go beyond what it was able to do efficiently. This fall I was actively researching other products to replace R:base as a development tool, until R:base announced that it was developing a new product that was going to cut past some of the limitations and a third party called AI-Ware formally announced a product called R:Turbo, which is a Clipper [compiler] for R:base. So those limitation are now gone, appear to be gone, and I'm certainly willing to, and the clients are willing to, wait out and see if these things will work as well as they say they are going to work.

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

I judged it on profitability reports. I judged it on personal interactions with people from the company. I judged it on whether how they were in fact responding to both the marketplace and to input from users about pieces of the product that were not working well.

What would it take to cause you to leave your prototyping tool and move on to another?

Basically business. If business dropped off, if there was no longer an interest in the product, both in the training and in the development aspect, and if we... weren't confident anymore that we were developing products we could be proud of with clients... we would seriously consider moving on any kind of challenging application to something else, but we'll stay with it for now.

What features would you desire in a prototyping tool that are not currently available?

Well, some of them we are doing ourselves. One of the developers, guys working with me, had a wonderful idea. In any working version of a database that we deliver we include a menu item and a simple form for entering on-the-spot reactions to the system, sort of a user log... So that we could come in and quickly print out a list of all the problems they have had, as opposed to relying on scribbled notes and things like that. That's something we can do ourselves. As a tool, maybe I wish there was some way of generating quickly fake data, that would be useful enough to allow you to start printing some prototype reports. Right now, in our prototypes, we are limited to delivering menues and a few entry screens but not the reporting process. There are some tricks that we use, but they are not really satisfactory: it's obviously phony data.

What shortcomings do you see in the prototyping method?

In the way we're using it, the only shortcoming that I see is that I still think we're taking too long to develop the product. I'd like to see our development time cut. But when it's followed the way I described it, I think it works very well. Basically we get in trouble if we try and shortcut.

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

From what you're saying [to interviewer] it sounds like those people work on a fixed price basis... That's not the way we work and we will reject contracts that say that. I just won't do that kind of work, it's a loaded gun at the developer's head. I'd rather be as honest as I can about what I think it's going to cost, but document clearly everything we are doing versus everything that was originally asked for, so that if there are cost overruns they [clients] see very clearly that it's tied to things they requested, or, in a few cases, that we were just wrong in estimating how long it was going to take. But we do actual time billing, in fact we bill in phases; we deliver in phases and we bill when we deliver.

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

I think it certainly speeds up the process because you've just done one just like and all you want to look for are the differences, you understand the hard parts: you worked those

through on the last project, and you make sure it works the same. So it's speeds up your questioning, it speeds up your specifications. But I don't [know of] any connection with substituting: you will have to do a prototype.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

[We chose to prototype this project] because of its complexity, because there were so many pieces that the sooner we were able to verify that we were on the right track the better.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

[Not transcribed.]

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

[Not transcribed.]

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

[Not transcribed.]

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

It's a funny business because many of people who are in it come from such varied backgrounds. I don't have a formal MIS background of any kind; I'm completely self-trained. And there are many of us like that, and there are many of us who have a formal background. And I think the lines are going to continue to get increasingly fuzzy, because this stuff is just more available to the general public and anybody who has a decent math sense can grab onto it and use it. So... Sometimes I have a sense that there are techniques there that I don't know about, that if I knew about them, it might make my life a lot easier. Sometimes a get a glimpse of that when I talk to people who had a more formal background. I'm talking about process, the planning process, the programming process, that will speed up development of programs and so on... It feels sometimes inefficient; it feels like we ought to be able to do it faster.

EXHIBIT G

Transcript of Responses to Selected Interview Questions

Respondent 5

Would you now share with me your own operating definition of prototyping?

Basically, I would say, it is an evolutionary approach to providing solutions to business needs. And, in my definition, it also has to involve the user to be effective. one that occurs in a relatively short time frame, dependent upon having the kinds of tools that will make that effective prototyping.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

Involvement of the user is obviously one [reason]. Effective tools, such as screen painters, report generators, database systems, dictionary systems, knowledgeable IS [Information Systems] personnel that can also speak business... Knowledgeable in the sense that they understand their craft, of how to use those tools and make them... sing... yet they can also turn that around and talk to business people... My gut tells me that smaller systems have a higher success experience...

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

Same kinds of things in reverse [refer to response to success question]: inadequate tools, not having user involvement, IS [Information Systems] thinking they know the answer [and] trying to go off and do it. To say they're not going to be a success based on size, I'm going to abstain from that.

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

Several things. I think that, one, we have the tools that can do more than if you follow a classic system life cycle, and if you used the tools you can iterate through it. Two, you don't spend as much time going around and around. If you study some of things in the industry you find out that inconsistencies or errors in the requirements or specifica-

tion phase of a project usually total somewhere around 95% of the problems in systems and somewhere around 86%... of the costs are correction by the time you get to a working system. If in fact you have user involvement using the prototyping approach, you catch those problems earlier thereby eliminating a lot of the cost, a lot of the time involved in generating that cost, thereby you can deliver the things quicker.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

The same kind of reason [refer to response to time question]. If you end up having fewer major problems that take less time and less dollars, time is money, 'cause you've got people-involvement, then it's going to cost less.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

[Not asked.]

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

User-involvement in the design process. You see, I believe the real problem is primarily one of communication. Communication, as I'm sure you are well aware, is an incredible art. It is no science whatsoever. And, if you have business person who understands his business and has a sense of what he wants to do, what kind of information he needs and how he needs it presented, and then he goes to talk to an IS [Information Systems] person, who really doesn't understand the business, those two are talking on different planes. And if all they have is an initial design meeting, then three months later the IS [Information Systems] guy brings a spec [specification] back, a document about an inch-and-a-half or two inches thick, and says 'Here, read this over and tell us if that's what you think'. First of all, it's written in computerese... Second of all, it's two inches [thick], he doesn't have time for that, if he's good in his business. Therefore the communication falls through the floor, it doesn't happen. If you in fact involve the user in the design process, one, the user does have to learn some IS [Information Systems] kinds of concepts, two, the IS [Information Systems] person has to learn some business concepts, and they have to force themselves into communication, they have to learn how to communicate with one another and communicate their ideas. It's a natural progression.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

I think it can go either way. That portends to an underlying thinking about prototyping. The two variants on that are: one, prototyping should only be used to model or mock-up a system. Once you are sure of what the design is, then you go off and code it for real, in COBOL or some other more conventional approach. The second is you take an evolutionary approach, which is my thinking more, that the prototype over time evolves to become the production model. You can use mockup effectively to get the communication vehicle going... and it's very helpful for design user interfaces, and the flow of a system. But, it doesn't really give you all of the answers until they [users] see real live information, in my opinion. So if you recognize it for what you can get out of it and no expect too much, I think it can be useful. [So] I mostly favor the evolutionary side as opposed to the modeling side. If you don't carry it all the way through, I personally think that you lose some of the benefit of knowledge that you gained when you now have to take those, convert them into written specs, which you are going to give to a COBOL or FORTRAN or whatever coding team to go off and do. Now I have worked on some large projects... but I don't have a real good feel on the vary large, large programs. I think that there are some underlying pieces that need to be addressed differently, and I think that the recognition is just beginning to happen in the industry. The focus needs to be not application oriented, not point-solution oriented, but data-oriented. That means that prototyping, then, doesn't take on the significance that it once did as of a few years ago.

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

[Not asked directly.] First of all I think development of an application should not concern itself overly greatly with efficiency. Let's get the solution out, get it working, and then you can go back and you can do, in necessary, if it appears necessary and I venture to say that at least 80% of the time it isn't necessary, then you can target certain areas, and you can put measures on to find out where the inefficiencies lie. There are ways of determining paths through the code to see where the real load is. And, at that point, you target an area and you go in and if necessary you write it in Assembler or write it in C or whatever you have to do, to go around a piece of it. But for the most part, I say, it is blown out of proportion; never do it first shot.

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

I guess I try to look at, one, the quality of the product. I guess the quality of the product is the first issue with me. If the product is good, then I think it will be around, it will be around for a while. Whether or not it becomes the superstar of the industry is another issue: not all the superstars are the best products. Number two: try to understand what they are trying to do with their marketing, where they are trying to go with it, the product. Number three: just some background on the company itself.

What would it take to cause you to leave your prototyping tool and move on to another?

Enormous amounts. None of these things [tools] are God's answer to the world. It always takes several times longer than you think it will. I've been around for many years and I still have difficulty getting my estimates on learning new tools and things. How many in the way of hours? I have no way of determining that: thousands of hours.

What features would you desire in a prototyping tool that are not currently available?

Let me take R:base [respondent's prototyping tool], for instance. That's the one I've worked with most recently. Better tools for managing multiple versions of applications. If in fact you're going to use it as a prototype, that by definition is evolutionary, and you're going to have multiple versions of applications, multiple versions of reports and forms, etc. There are no good tools whatsoever within R:base to be able to be able to manage that effectively. For me to do development here [his office] and easily port it to a client site is a pain in the neck. A central encyclopedia, by that I mean more than a data dictionary... R:base has a sort of data dictionary, but it's is not always easily accessible and you can's always do the things you want. Oracle's is a little better in that respect but even it doesn't give you straightforwardly "where used" information. It doesn't capture, for instance, what reports use certain data elements. The encyclopedia need to be extended to the point where you can do information planning, strategic information planning, right down through the development of normalized data models. Right now, the data dictionaries that are in all these products at best are nothing more than application-oriented or database-oriented dictionaries containing data elements, relations and some of the associations in some cases.

What shortcomings do you see in the prototyping method?

One [shortcoming] that occurs to me is the psychology. It's real easy to say that this is the end-all, that it will solve all of our problems, and it's not. It's like anything else, you have to do it diligently, carefully. I do believe it requires, and I ask people who are willing (and willing is underscored there) to learn and understand some of the business issues - to get out of their computer world and get into the business side. And I think that is a trend we are going to see. And it requires a user who is willing to go the other way as well. The issues to me are more people issues than they are technology.

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

[Not asked.]

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

No... If a developer, a programmer, an analyst has experience then what you are saying is, because he did this before, he now understands all the business needs. Not true... It might be a similar application, but the user has specific needs for that specific area of business. And that may be different from what that developer did before, and he won't know it. And if he doesn't talk to the user, he still won't know it.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

Primarily because we needed to really be sure we understood what the user's needs were.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

Involvement of the user is the key there. And you are much more in a training role than you would be in a standard analysis, systems life cycle kind of approach, in that you've got to get the user involved, explain to him why he has got to spend his time there, help him understand the processes involved and actually selling him on the benefits of him being involved. So it is much more of a P.R. [public relations] kind of role than the normal.

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

I never thought of it either way, as a matter of fact, come to think of it. I'm just not terribly anxious one way or the other; I've got a job to do, I do it... [I do find myself more eager]. If I've got to do something a standard old way I'm not nearly as warm on it, [because] I know it is much more tedious. Like anybody else, I like to see benefit from my activities. Delayed gratification is fine when you have to do it, but it isn't always necessary, and prototyping is one of those areas.

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

Prototyping should result in a smaller development team, therefore the IS [Information Systems] person should be providing the leadership role, the drive behind the execution. [Asked if this conflicts with desire for more user involvement:] I think it's part of the evolutionary process. The users have not been involved. So, you can't just one day flip a magic switch and say, 'OK, not it's your turn to take over responsibility for all these things. You tell us what you want and we'll go do it for you.' The IS [Information Systems] has to take on the leadership role to sort of sponsor getting the user involved until they start taking it out of our hands. We want to get them to clasp it to their breast and say 'Wow, this is neat, I can control this, and I really got what I want this time', instead of being something this.... You have to introduce them [users] to it.

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

It depends on the person. It's totally dependent on them. Most often, they want us to take on the lead, but I think that more speaks to the personalities of people.

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

The one thing I did want to be sure and bring out is an emphasis that prototyping is a useful methodology, a useful approach to developing system, but I think we need to keep in mind that one of the key problems that we have in the IS [Information Systems] [Information Systems] world today is that we are buried by backlog, both the visible and the invisible, the ones that they never tells us about because no one will ever get to it anyway. Most of the changes in methodologies that we have employed, for instance structured

coding and structured design methodologies, have been useful and have seen some productivity increases, but according to the numbers I have read that have not been large, say 10% or so give or take. We are probably not going to make great leaps in terms of our own productivity, in terms of our overcoming these kinds of backlogs just by making evolutionary changes in the methodologies and approaches we use: better tools, better languages, prototyping versus the standard systems life cycle, I think that will have a positive impact. But it is still very point-solution oriented. I think that we need to look beyond that, I think that we need to think more broadly, I think we need to let our minds go, start thinking creatively about some of the things that we can do to make major leaps in terms of providing information, and there are several things that occur to me. Number one: changing the whole nature of the use of computers in business, changing the nature of the way IS [Information Systems] does its job, what that job really is, classically IS [Information Systems] has been the full providers of information... and they [Information Systems] accepted full responsibility for the validity and the quality of the data. I think that's probably an error. I think that we need to put some of that responsibility back where it probably belongs [with users] and that's not an easy thing to do. People won't take on responsibility voluntarily, there has to be some benefit there. Users need [to take on] responsibility for what the data is, the definition. We need to end up with corporate-wide... standard definitions... I think that we need to recognize that there might be other ways of doing things in terms of how we gather and maintain information, using such things as subject databases on a corporate-wide basis, taking a data-centered approach instead of an applications-centered approach. One of the reasons that things haven't worked real well is when you develop a series of point solutions that talk to each other a little bit, they would pass data back and forth, then you end up with data redundantly in several applications, often inconsistent over time... If we instead take a central database kind of approach, even though it might be distributed, and then you have applications feeding off of a central point, you may eliminate a lot of the problems. And, instead of just seeing minor improvement of 10% we ought to be able to see improvements of 5 or 10 times. With prototyping itself you can get large improvements... just in the application development phase. But I think you can make an awful lot more when you totally eliminate the big part of prototyping, which today is the definition of data and developing of the data structures and populating those and developing all the interface tools to maintain all those. You eliminate that with a central database capability and the tools surrounding it. I don't want that to get lost while we're focusing on prototyping.

EXHIBIT H

Transcript of Responses to Selected Interview Questions

Respondent 6

Would you now share with me your own operating definition of prototyping?

My definition of a prototype is a vehicle of illustrating to the user the end result of a developed system. Basically you are giving them a superficial view of what the system will be. I contrast a little bit with your definition, or maybe I just didn't understand your definition, I don't really see it as being a rough draft, but more as 'This is what it may look like'. A lights and mirror show, OK? From an online perspective, you might show them a few screens, some database access, some editing capabilities. From a batch [perspective] you may throw a few reports in front of them and so forth. I don't think the guts of the system are necessarily going to be present. When I say 'the guts' I mean the actual extensive calculations and so forth.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

I would say that they are relatively user-friendly. From a visual and understanding perspective, they are user-friendly. They are larger systems. The calculations and the internal process is relatively extensive. That's with my experience.

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

A lot of follow-up maintenance, delay in deliverables, user dissatisfaction.

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

Because... the key thing in successful system development is a complete understanding, from the user's perspective, as to what the system is going to be doing. They're the experts on the application, the programmers are not. And, if their feedback and input is not involved extensively then you are not going to get a system that is replicating their knowledge. The vehicles for doing that: one of the vehicles is

proper analysis and design methods; another is prototyping, giving them a light and mirror show as to what they are going to be getting and giving them a chance to say 'No, you're way off base' or 'You're close' or 'You're right on'. Although a lot of time is spent doing that, and the fear sometimes is that we're going to spend all this time and end up with nothing, whereas what you can't impress on people [enough] is that if you don't spend all that time, you may still end up with nothing. And you may spend a lot more time trying to pick up the pieces. And I think the key there is not having to pick up the pieces at the end. You really have to take a look at what you're talking about as far as a deliverable, and if a deliverable is a successful product, then the time is definitely shorter. If you're talking about throwing something into production and then cleaning it up for a year because you did a horrible job, then...

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

I really see prototyping as a tool during the analysis and design phase, as opposed to not using it at all. And, as a comment as to prototyping too, depending on how you define prototyping, even if it is not a lights and mirror show, during your analysis and design phase you may be doing a lot of verbal prototyping... but still there's a lot of conceptual and verbal prototyping that goes on. I think once you do put it on a tube or do put it in a report format it's going to help even more...

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

Again, prototyping a system is showing the users potentially what they are going to get as a final result. It gives them the chance to actually see the system in a pseudo-production mode. If they don't like it, hopefully they will tell you right then and there that they don't like it, as opposed to maybe just doing a straight paper analysis and design, and then giving them what they're seeing... Through the development effort of a system, user participation continues not just from analysis and design but from a testing standpoint. A lot of time has to be spent on user tests to verify results and so forth. I'm not sure but... to a certain extent I wonder how much prototyping may be done even at the system test point. I don't know if that's the best place to do it, but I think a lot of times that's where it's done...

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

I think that initially in systems design, the paper methods are a better way of getting at user's needs. You have to talk. I think if you show them too much from a prototyping standpoint you may sway them away from what their actual needs are. Whether that might be the correct way to go in the future is one thing but really the analysis and design stuff is to find out what their needs are, what their requirements are and the way they do their business now. One of the results of a proper analysis and design phase is not only giving them what they're doing now but also suggesting a better way to do their business without being to snotty about it. So I think prototyping has has be introduced at a certain stage, or in a certain way, so as not to sway the users away from what their actual needs are. I could see it as being a potential danger, too.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

I guess it depends on the application. If it is strictly a data-collect information system, then I see no need for developing a lot of computations or internal processing... You want to show them what they're going to be entering and what some of the reporting functions are going to be, and that's enough. Even as you grow in complexity as far as calculations go, depending on the application, I think to a certain extent your prototyping method at that point is going to be on paper. You're going to sit down with either a dataflow diagram or analyzing relationships between entities and you're going to determine what the processing is and then you're going to take some test cases through. I think really what you have to do is to take a spectrum of applications from a very complex financial or engineering application down to a simple data-collect system and analyze the amount of internal processing that should go into the prototype. As the application grows in complexity from a calculations standpoint I think the internals of the prototype should grow as well, so there's a relationship from that standpoint.

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

I think if you spend too much time putting the internal processing into the prototype, and to do proper coding from that point, it would be my hope that the code that was used to prototype would be structured code. If it's not structured code it's not good code, and I wouldn't want to use it for final system development, simply for maintenance reasons... Part of systems analysis and design, one of the very first steps, is to define the scope... you've got to define your scope and take it a piece at a time. You've got to limit your scope to something that's doable in a good time frame for the user. From that standpoint, once you've defined that scope, you've limited your users to a certain discussion point, as to what you analyze and design. From that standpoint, it's up to proper design methodology to come up with an efficient system. Efficiency doesn't come from how you code it, it comes from how you design it.

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

I would say by their [the tool's developer's] client base; by the appearance to the tool that we purchased. We don't purchase a tool unless we've seen it used, had it demoed and so forth. So it would definitely be as a result of... you can tell whether it's sloppy or not.

What would it take to cause you to leave your prototyping tool and move on to another?

I guess that just from a practical standpoint, it that's all a client had, we'd have no practical choice, in a mainframe environment. I would say just... ease in use, quickness in delivery.

What features would you desire in a prototyping tool that are not currently available?

I would say stronger editing capabilities, perhaps a quicker way to give them more internal processing... Perhaps one of the reasons I like to limit my prototypes to a lights-and-mirror show is because I don't want to spend a lot of time putting in a lot of processing that a lot of users on face value may not care about. But if you could do that, and even from a marketing standpoint if you could do that, you probably would impress a lot a people, if you could prove to them that the systems is really going to do what they want, it's not just going to show them what they want, but it's going to do what they want behind the scenes.

What shortcomings do you see in the prototyping method?

I guess the limits, based on what I was just talking about [see response to features question, above]. From an online perspective, prototypes are nice because you can enter some data, and the person who's doing the prototype or using the prototype enters some data, and so forth. I haven't seen a lot of tools where you can program some decent edits in there, to prevent... even cross-field edits or even cross-file edits, that type of thing. Let the user sit down and play around with it, so if they enter something that isn't in sync with another field, it will tell them; that will impress them more. Even from a reporting standpoint, too: a way not just showing them a report, but actually producing a report right then and there. And those may be available, and I'm just not aware of them yet.

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

Actually, I guess I've never really given it [relationship of prototyping and financial operations] a lot of thought. I don't know if we use it [prototyping] in any kind of formal manner. Prototyping in its general definition is probably used in a conceptual way. We visualize what is going to happen if we take various actions and so forth. In some cases, we visualize it on paper, but I don't think we actually ever use prototyping as far as making decisions out of this office. Maybe we should. [In response to a probing question with respect to clients and the role of prototyping in business operations:] I think if the client has agreed to use our services, and we use prototyping, I think the impact is a more professional appearance for our organization, that we really know what we're doing as far as design and analysis. It promotes us as an organization that's not trying to blow something by a client but instead is giving him [the client] the chance to say 'yea' or 'nay' to it. Prototyping is a marketing tool. I think the plusses there are pretty obvious: if you can show a client as much of what you're trying to market to them as possible, they are going to get a better appreciation, will understand better what you're delivery and will understand better, perhaps, the estimates that went into that delivery date. [In response to a question regarding cash flow:] It depends on the actual definition of prototyping. The way we do a lot of our systems development is in phased implementation, or at least in phased deliverables. You're going to deliver a certain phase of a system and the user is going to be responsible for testing that phase, and they you deliver another phase... So I suppose in a way the definition of prototyping is taking the actual system and saying 'All right, this portion is done, test it'. And the user is going to tell at

that point whether or not they like what they see. So, from that standpoint, the user can see progress behind a system and they can appreciate it more and they can probably at the very end be very familiar with the system and be ready to go. So, I guess I never realized how general prototyping could be or whether it's just specifically taking a tool and prototyping a system. I suppose you can really generalize it as far as its use.

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

It definitely could substitute for prototyping only because the developer and the user are going to be on the same level at a quicker period of time. I guess in a way it might even be good if it does substitute for prototyping because the developer puts together a prototype, the users are still the experts. One of the problems in today's programming market is that there are a lot of programmers out there who get to know their application, get to know their business pretty well, and start to think that they can tell the user the way they should be running their business. And some of their suggestions might be good, but to a certain extent they have to temper that. If they push those suggestions too much via a prototyping tool, I guess my fear would be they would push the user down an incorrect path.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

Number one, it [prototyping] was available. Number two, the users were extremely computer illiterate and were too familiar with a very antiquated system. They were being introduced into an online environment for the first time in many, many years. And the prototyping tool was sought as a way to bridge that inexperience over to comfort of the system.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

[Not transcribed.]

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

[Not transcribed.]

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

[Not transcribed.]

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

[Comments made toward beginning of interview.] I'm a strong advocate of an extensive analysis and design period during system development... Supposed at least 60% should be analysis and design. I strongly agree with that. I've seen situations where that has been the case, and how successful the system turned out, and I've seen situations where that hasn't been the case, and the resulting mess we ended up with... I think the impression a lot of times is that's a methodology [traditional full analysis and design] should be used in the mainframe environment, in the mini environment, but in the micro environment, if you can get something quick and dirty, you can do it quick because it's a small machine and you have more control. Some of the largest mistakes made in the micro area right now is that that same methodology isn't carried forth to the micro world. And, because there is more control and more volatility in the micro environment, you have to apply that methodology even more so in the micro environment. And I think that's one of the largest mistakes made in micro development work now is that not enough... time is spent on really finding out what the user needs, especially with users who are relatively computer-illiterate... I see prototyping as being one of the steps in doing that. [In response to request for additional commentary:] I think successful prototyping depends not only on the people doing the prototyping, and the user's acceptance, the user's involvement with the overall design, but also depends on the tools that you are using. And if you have a proper tool, the prototype could be a real plus in the design. If you have a poor tool, it could hurt you very badly. Again, I'm a proponent of analysis and design methodology... Prototyping has to be introduced at the proper time and shouldn't be used to push the user towards, I mean you have to push the user a little bit, but you don't want to push them too far down an incorrect path without giving them a chance to fully explain their disagreements or agreements with various components of what you are showing them. Again, I'm a very strong proponent of user participation in the whole development phase, and perhaps even in the actual creation of a prototype, the user should participate in the final result, as far as a prototype goes. ...From the a business standpoint, they [users] know their business, and nobody knows it better than the users, although there are a lot of programmers who feel they do, but they [programmers] don't. One of my fears, I guess, is that if you show too much flash to the user, you don't want them to be sensationalized by the flash as opposed to what the system should be

doing. You have to tender what you do. I think successful prototyping depends not only on the people doing the prototyping, and the user's acceptance, the user's involvement with the overall design, but also depends on the tools that you are using. And if you have a proper tool, the prototype could be a real plus in the design. If you have a poor tool, it could hurt you very badly.

EXHIBIT I

Transcript of Responses to Selected Interview Questions

Respondent 7

Would you now share with me your own operating definition of prototyping?

I have never used the term [prototyping] because I've never really been with a lot of programmers, I just take a sort of common sense, no nonsense approach to the problem a client may have. I'm self-taught on computers, which may be good or bad, so there are a lot of terms I may not be familiar with. ...You're [to interviewer] looking for a definition of how one attacks a problem? And I do it basically by putting up dummy screens and dummy reports, and it sound very close... If I had to come up with a word, maybe it might be prototyping, from your definition, it's not that far off. I find that most of my people, now usually I'm dealing with either managers in MIS or department heads or the vice-president of a corporation or something, they can usually tell, it's like pencil selling, just looking at the picture of the screen on a piece of paper or an output report, they could really care less as long as it doesn't consume a lot of their time and it's a rather easy-to-learn and easy-to-use application. They can tell from looking at the input screen and output report whether I'm getting what they want.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

That stage from getting the commitment to go ahead with the project into at least the debugging phase, that is putting it online and trying to start the engine, is shorter.

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

[The system becomes] a workhorse because [the developer] hadn't fully understood, and the client wasn't capable of, or failed to tell [the developer] a lot of things. He [the client] was organized in his own business, and had [the developer] gone further with prototyping [the developer] would have uncovered that.

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

It shortens that time, it's a good vehicle to handshake with your client on, saying 'Yes, this is what we want'. Once you know your design, coding is the very next step, and it really saves on that because you don't have to then come back and do a lot of patchwork, boilerplating on the code.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

It would follow [see response to time question, above]. I would have to assume so if it...

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

[Not asked.]

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

I think the prototyping does [elicit user needs more effectively]. Flowcharts can be a situation where you can't see the forest for the trees. I think the prototyping, at least the way I handle it, just showing the input screen, the menues and the output reports. It's my job to make it an efficient process in between those two.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

I don't think they [prototypes] should [contain underlying computations] unless there is something very involved, very unusual about it. Most people understand that the math is going to work, or the logic, or whatever is the guts of that system.

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

I don't agree with that [need to recode prototype for executional efficiency]. If your design is correct, and I mean by 'design' that you have understood the application well enough to know exactly how index everything, which is usually where slowness will hit you in the face, then it follow that if you index properly, your application... The whole reason for indexing properly is so that your application will run in a reasonable real time.

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

I borrowed the confidence [in Progress' {respondent's prototyping tool} longevity] from another person who talked me into it. He hasn't been wrong yet. That's not the most logical way at arriving at that [conclusion].

What would it take to cause you to leave your prototyping tool and move on to another?

Nothing [would block me leaving my prototyping tool] if I could be convinced it [new prototyping tool] is a more efficient way, I can write an application quicker, from beginning to end...

What features would you desire in a prototyping tool that are not currently available?

I guess some rather sophisticated code generators. Maybe that's getting beyond the prototyping, but I could see that in a package. If I had all the knowledge in the world, and all the understanding, and wanted to write some tools to help programmers, I would try to come up with something rather sophisticated that would use standard blocks of code to accomplish certain tasks. Maybe that's a fifth generation language.

What shortcomings do you see in the prototyping method?

I don't see that it [prototyping] does fall down. Obviously, I think it's the best way to go or I wouldn't be doing it.

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

[Not asked.]

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

I don't know that substitutes but it makes the prototyping a lot quicker. The fact that he [the developer] has an understanding of the basic application going in [to system development], it's never going to be an identical vehicle that he produces, but it's going to cut down that time involved in prototyping itself.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

We came in to rescue a project on a short-term basis and then made a decision, seeing what else is available in the industry [hotel property management] and what it cost, to go ahead and develop our own. So, to develop our own, we are prototyping to, and for, a select audience, and trusting that that audience we selected will be representative of the market we want to address.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

All I'm trying to do when I meet with a client is understand what he needs, and, then apply my experience on how to best get it from input to output. It seems like such a simple thing and we spend all this time doing it. I don't want to give him a dissertation on the way computers work and all that. I find if I can just lay a few pieces of paper on his desk and he has a chance to really look at them... and if I warrant to him, based on my experience, that if he accepts that I can produce that and I can come up with a dollar figure for him. That gives me confidence going in, it's like leading somebody through the sale. It's tangible...

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

[Not transcribed.]

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

It's funny: as well as they usually know their businesses, they aren't necessarily logicians. I find I tend to be the one who brings up alternatives and in many cases changing the way they perform some task even manually.

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

[Not transcribed.]

EXHIBIT J

Transcript of Responses to Selected Interview Questions

Respondent 8

Would you now share with me your own operating definition of prototyping?

Here [a large consulting firm] we... have three different ways of looking at prototyping. One is DBMS [database management system] application development... I define it as being specifically 4GLs [fourth generation languages], very advanced application development tools. Prototyping for me solely means an iterative process where the users get to make dumb comments and you respond to them and spend more of their money and you make the product better... The other two types of prototyping... are... I have called it real-time graphics simulation... generally [meaning] maps... The prototyping part of that is to show somebody what it looks like before they actually develop the system. It's is very related to application prototyping in that the users are involved and users do get to make their changes. The big difference is that, when you are doing application prototyping, you are doing half the work by prototyping anyway. When you are doing a military [map] simulation package, there is really not that much work involved... They want to be able to see up-front 'Is it worth doing this?', and it's not really that iterative, 'Let's improve it'. It's just get something before you spend [a large amount of money]. And the third type of prototyping we have is mock-ups, hardware mock-ups, physical mock-ups of terminals, keyboards, even going to screen displays and things like that.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

An extremely solid design to begin with... You've got to come in there with a system that really seems to do the job the first time around, so that they [users] can understand it and start improving it. Two, for it to come across as a successful contract when it is prototyped, there needs to be a billing structure in place or an overall final completed price that is lower than what the client secretly was prepared to pay or it doesn't matter how good the system is, you had an unsuccessful contract because you spent too much money improving the product...

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

One, coming in with what looked like a complete design and a working mock-up for the users to improve, that didn't come close, that missed a significant part of the system. You can never prototype it or modify it to work, you just never get their involvement, you never end up making it, you patch the thing together and you are never able to start over again and really make the system. Two, ...the client paid you minimally to show them what the system would look like, rather than paying you to develop it. And we spent an absolute fortune... tinkering with the design until the design was finally perfect, and we didn't get paid for any of that time. And we then had a perfect design that is truly 90% of a working system, and they [the clients] go somewhere else and have somebody else write the system.

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

It shouldn't. It should take longer... using prototyping. Because the harsh scenario is that if you don't involve the user and you don't do prototyping, you go away in a closet and produce the system as rapidly as you can and you say that it's done and you deliver it. And they [the users] live with and they come back and pay you to make some changes but they weren't involved. You took what you thought you understood [was] the system and that should be much faster than where you juggle it with user involvement and maybe you work on another project at the same time, and you let them keep for a couple of weeks after each iterative step, where they take over a piece of it, and that one [the prototyping approach] should take a lot longer. The reason I say 'should' as opposed to an absolute there, is because in some cases when you have the user involved and you're prototyping you just naturally stay in closer contact with the user and they know what progress you have made every week. In that other box, when they're not involved, you should be... able to say 'I'll sit down and do it in three weeks and we'll just do what we do and they'll just live with it'. But, in reality, since they're not around, you can spend three months doing hardly anything on it, before you finally reach a point where it's an embarrassment and then you jump [and do it very quickly]... It ends up taking longer that way just because you ignored the user.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

You do a better job, you give them more of what they want... [And that costs more] if you do it right.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

[Not asked.]

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

You need to have a combination [of prototyping and conventional analysis and design tools]. ...You need a good traditional design underneath it before you can start getting meaningful feedback from the user. But in terms of ranking the relative importance, I think prototyping is a way of truly eliciting user needs counts for 80%. The people who come out and say 'We can do the job. Let a traditional designer come in' are the people who have been giving only half of what they [users] want for decades and have every intention of continuing to do so.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

It's a question of tools for me. I understand and know that there are two different definitions of a prototype, depending on whether it is evolutionary or functional or whether it is just a mock-up. To me, it is entirely a quality of the tool. I have yet to see the non-functional type of tool, the demo program type [simulation tool], that is good enough for what I try and do. By the same token, the tools I use, the 4GLs [fourth generation languages], they are so good that in the same amount of time that it takes for me to generate just a working menu structure and some screens, I've also done 60% of the underlying work on the system. Since I have good enough tools to do a functional prototype, it is certainly worthwhile to do so. ...And that's made even more important by the fact that I don't have good enough tools to do a non-functional prototype that works.

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

It's entirely [a question of] the tool. You ought to start out with a tool that's going to be able to do the job, that's going to be able to complete the final system. I have this bias, but I think again that the people who say from the other world that (it's all part of the same argument) 'It costs too much money, it takes too long to involve the users, it takes too long, it doesn't do a better job, our designers can do it better, if you bring them [the users] in prototyping gets drawn out, and even at that I create a working prototype and then I throw it out and I sit down and write it in COBOL'. I think that is entirely an obsolete attitude that is based on protecting one's position. And the priesthood in control of the magic of computers rather taking advantage of some tools that are out there, the ability to rapidly produce and modify a good system that really does do what the users want and not what the programmers thinks is the fanciest, neatest, most technologically sophisticated way to do it. ...[Recoding for executional efficiency] is the wrong attitude. It the attitude of a computer priest who is talking about numbers that are not relevant to the user as means of making it clear that they're the only people who can work the system. That depends entirely on the tool. You can certainly use tools, dBase, Revelation that allow prototyping but are so slow that unless it's a pretty dinky little application when you're done, you're going to want to throw it out and redo it in lower level languages. But there are plenty of alternate tools that are much better... [Regarding those opposed to 4GLs:] I can understand that attitude four years ago [circa 1983]... but the same thing is true all the way down to the assembly level: the thing will be much more efficient if you write it in Assembler. Processor speed has increased in my world, the micro[computer] world, so dramatically that if you buy the right tool... tools that support prototyping, good 4GLs as opposed to popular one aimed at end users...

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

One, the least [important consideration] is the quality of the tool. [One] tends to believe that the quality will keep them around for a while, but I've seen too many disappear despite their high quality. The second is the ...operating system environment. If I think they're in the right environment, I tend to give them a good chance of hanging around longer. And third is the extent to which they are actively supporting the development of vertical market applications or development of applications in general and support of dealers who are writing those applications. ...[If the tool

developer] is just selling the language, rather than being supportive of developers, they're all going to die because none of them are going to be dBase or Oracle or Unify [sic], which is a shame. On the other hand, if they are actively support their [vertical market application] dealers, I think they've got as many years guaranteed as I care about. And that really, definitely means actively supporting dealers who are writing applications, it doesn't mean retailers of the language, or just consultants who will show someone how to use the language. If there aren't people out there seeding the market with applications written in this product, then there's a real question as to whether they're going end out with a large enough installed based and large enough visibility to justify working in it.

What would it take to cause you to leave your prototyping tool and move on to another?

I'd leave any one of the tools without hesitation if I had a client who was willing to pay to have the system developed in a different 4GL [fourth generation language] that I knew could develop it, the final functional product. I wouldn't care how good the prototyping features are of it, I'm really not attached to how it prototypes... All I care about is that ...it's a database that I can develop my application in and I've got a client who will pay for it.

What features would you desire in a prototyping tool that are not currently available?

Every feature I care about is available in one or another product, but no product has the combination of the ones that are most important to me. ...Each thing that I can think about in Progress that I really need, or in Helix that I really need, is something that either the other has or Revelation... one or the other of them has it, just none of them have it together.

What shortcomings do you see in the prototyping method?

...[Prototyping] is the right way of bringing experts in... to get the correct product, the product that will actually do what we need it to do. And by and large, people aren't willing to pay for better products. Prototyping to me is not aimed at being a more cost-effective or a more tightly disciplined approach, it's aimed at producing a better product. And there is not a demand from the market... there is a demand from the market for a better product, there always has been, but it's not followed up with money. They're never willing to pay to get that better product. From the other side, from the general developer side, there is virtually no desire on the part of most developers to come up with a better product. Both sides seem willing to complain about delivering mediocre solutions at a moderate cost.

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

[Not asked.]

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

It [previous development experience with a similar system] can completely substitute for the system design, so that you can come up with that good basic system that the users can monkey with. But it's not at all a substitute for the prototyping. By 'prototyping' here I don't mean the creating of the first prototype, I mean for the actual process, the iterative process that involves the users. The whole reason that you're involving the users is to find out if your assumptions are correct, and to find out not just whether or not your product will work in their environment, but if they have got particular quirks that will make them happier with one type of menu as opposed to another, or one type of screen placement as opposed to another. Whether, to make them happy, you got to get off of the way that you had done it before and let them [the users] try it and tell you what works for them and doesn't work for them.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

One, the customer got burnt before, when they previously did this [unintelligible] on a PC, the system did not serve their needs. Two, Helix just happens to be, since I wanted to use Helix anyway because of the multiuser features, and because I think it's a good tool for developing applications. As long as you're using Helix, you've got to prototype, you've got to give it to the user in pieces and let them make changes so they they can feel comfortable with the tool you [the developer] chose... And three, because of that one history report [previous described in interview but not transcribed], it's such a flakey system. It's the same reasons why, when I rank those [meaning this system] on that numeric scale, it's an inexpensive system, it's not big and it really shouldn't even be considered all that complicated... [but] it [history report] is one of the most complicated, one of the most convoluted ways of tying information together that I've ever dealt with, despite how small and cheap the whole system is. And I have to have them prototyping it or I'm going to get killed at not quite putting the pieces together correctly, something that could not be achieved solely through system design... The design

structure underneath is fine, but the actual surface that they see, I just couldn't get it right by myself.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

The truth is more anxious... It should be less because I know the end product will be better. The reason it's more is because I know the costs will be higher. When I'm involved in prototyping, I'm constantly scared that they're [users] not going to understand why the cost is going up, despite the improvements I'm making, and therefore I'm working harder to hide costs, to cover it at lower costs, to put in extra time that they're not billed for, to do everything I can to try to keep the costs as low as possible, and that puts inordinate pressure on me [as developer].

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

[Not transcribed.]

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

You [the developer] have to do it [present system options] and you have to live with it.

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

Most of my thoughts on prototyping come down to that duality. There are the critics, and I tend to not agree with their entire rationale, and there's any number of little details that you can fight over, 4GL [fourth generation languages] versus 3GLs, prototyping taking too long, should the users be involved at all, how important are design groups, and all of that to me are little details that mask the true problem of one world that believes that they are the only people capable of producing a good application and don't seem to care about the fact that that user may not be able to use the application, that might be a great way of tying together a couple of different kinds of paper information, but it might not be useful, it might not be tailored to the skill level of the users. Programmers are die-hard opponents of usability testing, we [respondent's organization] are big proponents of usability testing. The same

people who fight prototyping are the ones who will think 'I can come up with a good product and when it's done it's done, and it works'. It's very hard to get any of those people to understand that even though it might work, and it might be fast, and it might produce all of the output you want, the fact that it's got twice as many reports as it should have, or that it doesn't use the language that the user is familiar with, or that it doesn't match... that particular user has certain eye problems, or certain physical problems or certain ways that the user is used to work, it doesn't match that user at all, it matches the generic user, those same critics are opposed to it [prototyping]. I tend that prototyping is just one more way of producing a better product. The flip side of that is that the programmers don't want to do it, and the customer's don't want to pay for it... If you've already made the commitment to using a 4GL [fourth generation language], a DBMS [database management system] type language, then, I think they're [prototyping and conventional development methods] more or less the same [with respect to time and cost factors]. There's a difference in cost doing the prototyping, and I would guess that it's probably a 20% difference [prototyping being 20% more] to produce a better product... using the world of 4GLs anyway, we've already made that commitment. Now, we can either do it ourselves in our ivory tower, bring all of our vaunted analysis skills to bear, that can be shorter and cheaper by a factor of 20%. That's radically different than the other world, going back to the third generation language. ...[Within the 4GL world] it's slightly longer to prototype, and for some reason, you're on the spot in terms of cost. When you do a whole system yourself, it's kind of magic and come back and there's a certain cost, and they [the client] grumble and they accept it. You always have some figure in your head, you know if they are willing to pay fifty [$50,000] or if they are willing to pay fifteen [$15,000] or whatever it is, and you may end out a little over that, but in reality your time might have been substantially over that. You scale everything back to get it done as close to that figure as you can. As soon as you start involving them [clients], there might only be a 20% involvement but it drives you crazy. Every time you talk to them on the phone or see them in person, they make a dumb little change, and you've [the developer] done what you could to get them to make the dumb little changes up front, you've made it clear to them that it will cost more to make this change later, you get way into the process, and they want to make a petty little change that will cost you real time. You can get paid for that but you know they [clients] don't remember any more that it costs them more. They might make ten stupid changes which add up to ten extra days worth of work. And they remember all of that as being one extra day of worth of work. So now instead playing with a ceiling of fifty [$50,000] to try to get under, you feel like you should be able to play with a ceiling of sixty [$60,000], a 20% increase. But the users only up their ceiling to fifty-

on [$51,000], so the stress level is enormously increased, and it's all related to cost. ...It's not that much more to do the prototyping, but the stress level in the game we play with the client, and just the overall cost...

EXHIBIT K

Transcript of Responses to Selected Interview Questions

Respondent 9

Would you now share with me your own operating definition of prototyping?

It's a method of designing systems utilizing the end user in a much stronger consultative type of role so that the end user is, in fact, doing a lot of the systems analysis themselves.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

The more input and the more time the end user is willing to spend, the more successful the system is. It almost becomes proportional.

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

That's when the end users had little input and decisions were made on a hierarchal level, from administrators that really had no use of [sic] the system but knew what they wanted to get out of it.

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

Prototyping is definitely longer. ...there's a constant interaction with the end user and there's a constant revision of the system to meet their [users'] specific needs. And sometimes their needs change as the system develops. It's an easier task to build. On the traditional [methods], you can give a quote upfront and say, 'This is what it's going to cost', because you can kind of benchmark it. I think a prototyping system, it almost lends itself to a hourly type rate.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

Probably, I'd have to say from the time factor, a little bit more cost, but there could be other criteria that enter into that, too.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

I'd have to say they're [prototyping versus conventional development methods] of equal quality; I'm leaning toward greater quality [for prototyping]. Again, I'm talking from an end user's point of view.

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

In some respects, I think I'd have to agree that the traditional methods are more effective in getting at what the user's needs are. Prototyping, I think, addresses what the perceived needs are a little bit better.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

I feel that the actual computations... I feel it should be a full-blown system because, again... it's an involvement of the user all the way, so that as the system is being developed, the user is putting input in. Then, eventually, the total system is arrived at. [It] involves everything, computations...

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

I guess I'd have to agree [that recoding for executional efficiency is necessary]. The systems that I've done, thinking back on some of them, and we've had to go back after everything was said and done and figure out a way to speed up the system. We try to address it [efficiency concerns] in the building, and I suppose as each prototype comes along we can put more design techniques into it, but there are some times when you are faced with an after-the-fact.

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

I got involved with R:base [respondent's prototyping tool] because a friend of my was involved with Microrim very heavily, one of the regional managers, and I became familiar

with it [R:base System V]... I took it [judgment of stability and longevity] from him. Right now, ...I just try to read the trade journals and see what's happening with the company personnel-wise. I don't think the stock is a fairly good indicator of stability. I like to see what's happening personnel-wise and also product-development-wise.

What would it take to cause you to leave your prototyping tool and move on to another?

Significant improvement in relational database characteristics or a better method of producing end user usability.

What features would you desire in a prototyping tool that are not currently available?

One of the things that I would like to see for the end user is a natural language interface, on the idea of [R:base's] Clout [a natural-language interface], but more developed, so that the end user can produce their own reports, query reports.

What shortcomings do you see in the prototyping method?

First and foremost, I have to say time. I think the prototyping requires a great deal of communication skills, listening ability. And I sometimes wouldn't get into it without the investment of time that that takes.

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

I think the heaviest thing that I've done is that I've quoted and have done job proposals based upon time I've estimated and I have grossly underestimated the time. That probably might be a little bit of inexperience on my part as well as using the prototyping method, but prototyping does tend to exaggerate the time frame. On the other hand, I don't think you can adequately charge for all the time you have to put in, too, because otherwise the system becomes too expensive, so there is a point at which you have to weigh the actual cost versus a realistic cost... And so in some respects, yes, I think it [prototyping] has cost us financially. However, on the other hand, prototyping has benefited us in client loyalty. I have no doubt about that.

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

There can be some substitutions for prototyping in that realm. Perhaps that person would be better than someone [with no experience]. The only problem that I would see there though however, is that sometimes they [developers] can enter into the development stage with too many preconceived notions from past system. And I think ...the communications skills have to be there, because I think it's too easy to close off your mind from what the end user really wants.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

[We chose to prototype this project] because we want to sell this system to police departments in general and police generally tend to be very closed, and if they find that other police helped in the development of a system, they are more receptive to it.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

[I feel less anxious] because I get a better feel for the client. I think I can communicate strengths and weaknesses with the client. I can get a better understanding of their needs.

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

I would tend to see it less often, the leadership role. I think it becomes more of an equal partnership [between client and developer], I would think.

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

They [users] are looking to us [developers] for direction and guidance. Once the dialogue starts, then there's a give and take.

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

[Comments made toward beginning of interview:] ...When we use this type of method of building [prototyping], I find that the clients that we've dealt with have been... There has been a much stronger bond between client and consultant than, I think, if I came in and I dictated to them, well, 'This is how you want it and this is how I see it and this is how you're going to get'. I have run into the problem, though, in doing it that way. My time estimates have been way out of whack. That's probably my biggest complaint, not with prototyping, but with the way we've been approaching it... it's that our time estimates get thrown right out of whack. [Recorded at end of interview:] As far as prototyping is concerned... what I have found is that the more input an end user has to the system the more likely they are to use it. And the more they use it, the better the system is. It's to the point that now I'm getting into the situation where I'm trying to get as many of the people who will implement the system, even at the lowest level, involved so they get a feel that this is their system and as a result I have found that, number one, they're not as cautious of the computer, as somebody who come in with other canned programming, and also they're more apt to take some experimentation with the system and try and get it to do more of what they want it to do, which only in turn means to us building more into the system. So that's the biggest thing I see about prototyping. Yes, it involves a lot of time, but if the end user says, well if the product produced is very good, that's my concern.

EXHIBIT L

Transcript of Responses to Selected Interview Questions

Respondent 10

Would you now share with me your own operating definition of prototyping?

I haven't the vaguest idea. Where it comes up is in terms of what am I going to do for a client. And to me, what I use instead of the term 'prototyping' is 'a general example of what the final output is going to be'. And I try to do as little as possible of that, because my experience is, and this is a real circular problem, that the client doesn't know what they want until the project is done, which is understandable, but what took me longer to realize is that I don't know what the problem is until I solved what I thought was the problem, and then discover that I have to do it over again. So prototyping is sort of making a very rough guess as to what the final thing is going to look like... But, I don't as a conscious policy have a standard concept of prototyping... I do it, depending on the circumstances.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

Where prototyping helped... it was because they [the systems] were conceptually complex, not necessarily complex from the point of view of the application, but complex from the point of view of what was supposed to happen to it, from the end user's point of view. And that's where it's been the most help.

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

There the problems were more technical and did not have to do with conceptual relationships of the information.

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

The issue is,...when are you done? And to some extent, prototyping can help you get done faster, but it doesn't help you get to the first implementation any faster. ...Do you know why it's so hard? When is it done? The only way I can end a project, I find, is to train the end user to take

it over, because it's always going on to something new, something evolved.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

Yes, probably, I think it does save some blind alleys and some redoing things.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

[Not asked.]

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

No matter what you do, the end user doesn't know what the system is supposed to do until it's working, then you find out, it's not doing what you want it to do. And, this is not a problem of lack of sophistication on the end user's part, this is a fact of life. In fact, what is most exciting about all this is that you really don't have any conception of all of the power of your tool until you start using it. And then, you decide what the thing's supposed to do. And prototyping can help you a little bit on that but until you are really using it, you don't know what it's supposed to do. And I thought that as I got more experience, I'd be better at figuring that out, and it's not true. Because only the end user can figure that out.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

If it's easy to do, the more can you can do, the more realistic it is. But, you don't necessarily have to have the computations for it to be meaningful. I find in many cases, the thing you have the hardest time with is, say for example, the layout of printed reports, and what goes in there. And even if all the numbers are zeroes and all the names are 'John Doe', just seeing that and saying, 'Oh, we want to have something else in there' can help you. So, you don't really need the numbers. People are willing to believe that the numbers will get right eventually. But, I find that often it's a lot more work in the prototyping if you're just creating the format.

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

If you prototype in a language, any language even if it's not the ultimate language, it's probably all right...[If one uses simulation tools] you're may be getting into trouble. If you do things that look [like] the report... [After confirming that modeling is answer:] I address it [executional inefficiency] by resigning myself to doing every application three times. That is what I expect to happen ...maybe three times in the same [development] environment. I find on average it takes three times to get it right. And, recoding is not necessarily inefficient [refers to previous comments that recoding took 10% of original effort].

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

[Simply market share].

What would it take to cause you to leave your prototyping tool and move on to another?

[Client who wants development in a different environment.]

What features would you desire in a prototyping tool that are not currently available?

[Did not answer the question: could not think of an answer.]

What shortcomings do you see in the prototyping method?

The basic shortcoming is... that you don't know what you want until you're done, and then you discover it's wrong. And you can do a prototype to eliminate some of that, but until you have a working system, you really don't know how people are going to use it.

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

[Not asked.]

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

It [previous experience developing a similar system] doesn't help, because it's not the developer's experience that matters, it's the user's experience that matters.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

[I chose to prototype this project] because the client was very unsophisticated and there was one level of conceptual complexity that was very difficult to explain but very easy to demonstrate.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

My plan of attack is to give the end user enough of a system so that I can feel that there is a common agreement on what the system is supposed to do. So, it really is a function of the sophistication of the end user.

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

Well, a little less anxiety, because you can say, 'Well, I don't know... to do this now, but I'll figure that out later'.

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

[Not transcribed.]

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

I always assume that I'm going to have to take the lead in presenting systems options. That's why my business card still says 'Management Consultant'.

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

Well, there's one important principle that didn't quite come up in this [interview]... who is the leader kind of thing. The real person to do the prototyping, from my point of view, is the client. And that to some extent, you can help the client do that, but that's the person who should be doing it. And it's most effective when you can help them prototype, in other words, help them decide what it should look like. Now, you can say, 'Well, that's why they hired you to do, so how can they do it?'. Because my concept is

prototyping is, you want to know what's at the beginning and what's at the end, my job is to do the middle...

EXHIBIT M

Transcript of Responses to Selected Interview Questions

Respondent 11

Would you now share with me your own operating definition of prototyping?

What you gave as a definition [refers to interviewer's introductory definition] is very close to what I would say is the case. Prototypes for us... we don't often have the luxury to develop a prototype for a system on a speculative basis.... In many cases, to prototype, as you said the key-word being 'conscious', comes into being from what we feel to be the best attempt, and what we feel to be a complete approach to things, and they [clients?] would consider not be something that meets their needs. We would end up being..., what we would probably do is a prototype. Prototypes for us often come from a situation where a client A will contract with us to develop a particular product. We will see that client A's way of doing things with this produce may have a market outside that of client A, yet much of what client A does is specific to them and would not apply to client B, C or D. Prototypes for us are that stage after it has [been] developed for client A, when we try to cut away those things which are personality-specific to him [client A] and to add those things which we feel would be useful to others and then present it to clients B, C, and D. And of course maybe bounce some of these ideas back off the original client, who in many cases, not that they intended to do it, but in many cases finance the original development... What we consider to be a prototype is when we've taken it from the very specific, we've shaved off those things that are very unique, clearly unique, present it to others to see what things these like and what things they consider superfluous. [Interviewer questions about development of the original system: what is developed using prototyping?:] In many cases not... Actually, there was one [case] recently [that] we did very much in a prototype case, that was almost verbatim your [the interviewer's introductory] description earlier, but usually it's not. Our whole raison d'etre is to develop software that is highly unique, and I would say that we put a strong emphasis on the initial design phase which often pre-empts it as a prototype.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

For us, there is torturing, almost, over user interface. That's my own personal area of handwringing... It has to look good, it has to feel good, it has to be something that... I feel that the background algorithms, the efficiency with which you do things, anyone with half a clue in

computer design can do that, its secondary, you don't even think about how it works, we'll figure out how to do that... What we'll worry about is what it feels like for somebody to use it. Can somebody go to that and feel like they've accomplished something, they've done something, this has made their job easier, it's made them work smarter, made them do more than they had before. I feel that the program should be designed in such a way that it's self-evident how to follow it, this is something that does something. It's clear, it's never patronizing, which is something that creeps into so many systems, and people know intuitively how to follow any part of it... Consistency, a clean, easy-to-follow user interface, attractive screens and reinforcing somebody's skills without being patronizing to them.

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

...If there was anything that holds true, it's accepting blindly that... client A's way of doing things is the way that client A's industry would do things, lack of exploration of accepted methods in an industry. I think the other side of that reinforces back into what is successful, and that is... when we deal with a client and we approach a project, I think we go much deeper... and much further afield from the original project than most consulting firms would... A thing that makes it successful for us... is first of all how to talk to people, look them in the eye and talk to them and secondly we know how to solve things, be it ourselves or a product or a concept. And probably a subset of that, we know how to deal with people without [angering them] and there's a lot of [consultants] who don't know how to deal with people without [angering them]. And if the short-term gains are lost from that, I know [unintelligible].

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

With us it really isn't a factor, because the prototyping stage comes for us after it becomes a product. It probably causes it to take a little bit longer, because we are willing to put more into an original project, feeling that the cost of that will be absorbed at the other end, when it is prototyped and exposed to others.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

Again, for us, probably more [cost] because our goal is to soak it up at the other end [when prototyping the system for other clients].

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

By catching it at that stage when you have something that you feel sort of does the project, you've not gone down any paths very far. You can show them [clients] the feature that you think is a nice feature, but if that particular feature may not be something that the product does not hinge on... when in the long run you discover that that's an [unimportant] feature..., there's going to be a lot of 'pride of authorship' in that that, there's going to be reluctance to change it, and there's going to be a lot of personality-specific [unintelligible] to whoever developed that... If it's prototyped, then you [invest small amounts of time] and the [client] says 'I don't think I'll ever use this', fine, they're never going to use this, let's devote our attention to something a little more creative. But, otherwise you're not going to turn your back [on work created].

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

Both [conventional development and prototyping] are necessary. For us, when it reaches the prototype stage, those initial studies have been done, so the [user's] needs are there. I think it allows us to catch previously unseen needs which fell through the cracks in the initial thought process [design].

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

Our usual way of what we consider a prototype here, under our terms, which we might call a demo and can distribute to people... will have whenever possible all of the functions of a give system workable. However, there are key parts of it that are limited. I think the functions should be in there but limited in scope. I like to give people something they can use and really work with, but at the same time I don't want to give people something that they're going to use and not buy a final version from me. So, a prototype to

me may have a few features flagged out. I'd rather take a menu item off, and have it be invisible, than have it say 'Sorry, that function is not available', although I have done that...

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

I don't agree with that. I think that to just arbitrarily say that because something has evolved up through a particular means it is inferior to something that you take from the ground up... There are times when you have to take a look at it [executional inefficiency] and say 'This whole... My whole approach to this in this section can be redone'. But, I see that as evolutionary. To take it and discard it because it has been enhanced really doesn't, ...I don't think it serves any purpose.

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

We don't worry an awful lot [about the longevity of vendors of prototyping tools used]. You get a feel for companies when you deal with... I think we have to develop a certain amount of survivalist instinct here and not depend on outside vendors for our success...

What would it take to cause you to leave your prototyping tool and move on to another?

It's not cast in stone here, what tools we use. Each of us [refers to employees of consulting firm] have our favorites.

What features would you desire in a prototyping tool that are not currently available?

I'd say we have everything that we need.

What shortcomings do you see in the prototyping method?

If anything, it's [shortcomings] small. But if anything, it would make it very easy for you to put up a promise of something that would be much more difficult to deliver than you would originally think... It would make it easy to promise the moon.

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

No, it's akin to apples and bicycles [i.e., prototyping and financial considerations are different]. I don't see a connection. Oh, everything... all are subsets of what should be feeding into eventual profitability but I don't see it direct link here.

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

To a great degree, its a significant feature [previous experience]. [Interviewer probes by querying the ways in which experience cannot substitute for prototyping:] Just from a psychological or sales point of view, where you're trying to present something to someone, I think you would find yourself bogged down in minutiae... without getting the big picture and showing somebody something. That person [with previous experience] would have a tendency to dive directly into detail...

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

[Recorded prior to specific question on why this project was prototyped:] This is something that, first of all, we would have a very difficult time on our own trying to get a picture of... It has evolved both in developmental environment and in scope... and it has now reached a point where it is able to be shown. [After being asked specifically why developer chose to prototype this project:] To be able to present it in a format that would get it to more people in as short a period of time as possible with a minimum outlay of capital.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

[Not transcribed.]

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

[Not transcribed.]

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

No, they [clients] want us to [bring up system options]. until they get to know us. If they were at that level [users bringing up system options], they'd be doing it [development] themselves.

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

[Interviewer states that original development is somewhat based on prototyping:] It could well be, it could be the way we look at it. We consider 'uncustomizing' when we [present] it [original development] to [clients] B, C and D... [Interviewer probes on the financial aspects of prototyping:] ...I don't come from traditional academia where I would go through a structured design process, we don't deal with people on that level. So, it's not like I've done it that way [conventional development] for five years and now try and do it [develop system] kind of off the cuff, and find that people pay their bills any faster; it's [prototyping] the way we do it and people don't pay their bills anyway; that doesn't make a whole lot of difference. I think it's just a matter of not having seen it from the other end [conventional development practices]. I enjoy doing it, it's less a job than it is a labor of love. Otherwise, I'd have gone out and got a legitimate job years ago.

EXHIBIT N

Transcript of Responses to Selected Interview Questions

Respondent 12

Would you now share with me your own operating definition of prototyping?

I would say largely it's the same as your definition [refers to interviewer's introductory definition], except that I would perhaps see it as being a little more rigidly defined. What we would like to do with the product, before we come out with a prototype, and then seeing the program evolve from there as a series of bug fixes and enhancements.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

I would say those products that were prototyped too quickly respond to a user's need for a particular communications function, which is essentially what we do, we do communications software. Usually this will result in, very often, in rewrites of other software packages to bring about these new software packages. In fact, that's how about three of our packages came to exist: they developed from other functions.

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

At this time, no, I can't [answer the question].

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

Because in delivering a prototype to our user, the user should inherently have less expectation of the final performance of the product at that point, since he or she is more involved in correcting its function or enhancing it, and pushing its development in certain direction, that their expectations for the long-term end-of-product is diminished and that allows us to create a product more quickly and get it out in the marketplace and then concentrate on pushing the product.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

In the short term, I would say that's definitely true [prototyped system can be developed at less cost]. I'm not aware of any long term effects that that [prototyping] might have. [Interviewer probes for reasons for short-term benefits of prototyping:] Because we can produce a product very quickly, and start deriving income from it. On the long term, depending on how the product sells, it may or may not be of more benefit to us, in that it may wind up taking a significant portion of our development time and diverting us from other projects.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

[Not asked.]

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

I would think that prototyping is a better way [to elicit user's needs] because it allows the user to literally get in on the ground floor of the system with its developers. I see traditional design and analysis being a more removed process.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

I think it [the prototype] should include the computations because in order to get accurate feedback from your users I think it is better to have something more accurate for them to give you feedback on.

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

Usually through use of optimization packages to take the code that was originally inefficient and compact it down in terms of size and executional speed. Obviously, there are some things that [you] can't optimize with regard to the actual process that the program executes and in those areas, that will require some code rewrites, but by and large, I

would say, those elements of the prototype that are going to appear in the production system are coded well enough that they should not have to be written to such an extent that they have to be discarded.

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

That's difficult to say because we've been using, effectively, one set of compilers and languages for at least as long as I've been here... [tools used are treated as a given by respondent].

What would it take to cause you to leave your prototyping tool and move on to another?

[Not asked.]

What features would you desire in a prototyping tool that are not currently available?

I'd like to see from time to time, we do have some tools that allow you to analyze the structure of programs. I would like to see some more accuracy out of [these programs]; they often indicate too much and it's usually off on a tangent somewhere. So it really gives misleading results.

What shortcomings do you see in the prototyping method?

It [prototyping] allows... programs to develop haphazardly, at least from my experience. Very often you get caught in a trap of developing a routine to do something specific on one package, and then when you decide that you'd like to port it over to another package, it has already been written with its hooks into the other package so rigidly that you virtually have to rewrite it to extend that function to something else, instead of making something more [unintelligible] available, it's more of a private function, and that disturbs us.

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

[Not asked.]

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

I'd say no, because the prototype is really essential in getting from your initial concept to some piece of work in code that you can get out into the field.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

Because it started out as being an effort to bring a certain degree of teller automation to banks. We also wanted to give it enough expandability both to keep in touch with our competition and to provide tellers with the tools they need to get things done appropriately. And an expandable prototype really allows [one] to address those needs as they occur, whereas a fixed design and analysis, I feel, tends to limit our options.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

I feel less anxious about it because there is less short-term expectation of what the product is supposed to perform.

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

I would probably say less often as I find myself much more responsive to what a client is telling me with regard to what a package should or should not do rather than going into a particular and saying 'Well, this looks like this is what you should do'.

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

I feel that they [users] do bring up sufficient options on their own.

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

[No additional comments provided.]

EXHIBIT O

Transcript of Responses to Selected Interview Questions

Respondent 13

Would you now share with me your own operating definition of prototyping?

There are two types of prototyping. There's prototyping that's for the purposes of evolving a system. Then there's prototyping for the purposes of emulating an existing system on better equipment. Both fall into the prototyping category for a variety of reasons, but the evolutionary one fits your [interviewer's] definition perfectly. The emulative one is more common for companies of my type, in that we'll go into a situation and emulate, whether it's a manual system that exists right now or some offshoot of a mainframe or a micro-based system, also for the purposes of evolving. And I think the evolutionary process, when designed in general is effective if the intent is true evolution; that's what I would define as the basis of prototyping.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

Other than end-user involvement... Having a strong tool set that allows for rapid prototyping.

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

Generally, they are... existing systems that people attempted to generalize for use in a wider variety of areas, but never really met the characteristics needed for the wide variety of businesses that use them, which has got to be 90% of the applications out there...

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

The key is, if the prototype provides something that is immediately useful, then you can evolve from that point. If the prototype provides absolutely nothing that is useful...

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

If you take a look at the heavy initial investment we did in tools for prototyping, we probably have spent somewhere between $250,000 and $500,000 on those tools that we developed. And therefore, the first [application] system we produced probably came in over budget at about 150%. So, putting it that way, I'd say [prototyped systems are developed] at not necessarily at less cost, but, on the other hand, future systems that we do are coming in way under [cost].

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

[Not asked.]

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

Programmers for the most part find it very, very hard to use flowcharting. I don't think I'm different than anybody else, flowcharting is [cumbersome, not liked]. Flowcharting is usually done after the fact, not before the fact.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

Absolutely, include as much as you can [computations] in the prototype. You have to make a system marginally useful, or you're wasting your time... A prototype is not complete until it's a marginal[ly useful system]...

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

For most people, that's true [executional efficiency problems]. That's what makes us what we consider unusual or very good. [Interviewer probes reasoning behind this statement:] We can walk over other people in certain proposals in certain vertical markets just because of the skills we have developed [in dealing with executional efficiency]. We believe that makes us unique, and that's the kind of attitude that we have to have to be a small company and stay in

business over the long run, is to believe that we're actually better anybody else. [Interviewer probes for an explanation of these skills:] There's no such thing as true executional efficiency. You always give that up, there's always a trade-off between building something and having... Simon once wrote... that there are two issues with respect to systems design. One is efficiency and the other is effectiveness. You make trade-offs between the two, between efficiency and effectiveness, and we have obviously made those trade-offs. We think it is more important for a system to be effective than 100% efficient. And the way that's measured is: if in fact the issue of efficiency is computing power, the speed of most machines today are such that screens are returned almost instantaneously regardless of how silly a job one does in the efficiency area. If in fact a database appears to be updating instantaneously and the screens appear to be returned near instantaneously, then in effect you have an effective system.

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

Everybody takes risks - it's a risk... I guess there were two or three [reasons to choose a prototyping tool]. One is the skill of the players. Two is the amount of entrepreneurial money or venture capital that was put into it, and it was a good mix of both in a lot of it.

What would it take to cause you to leave your prototyping tool and move on to another?

I suppose them [Progress, respondent's prototyping tool] going defunct.

What features would you desire in a prototyping tool that are not currently available?

Other than some inherent problems or inefficiencies in the [Progress, respondent's prototyping tool] language, that they can clean up, the major issue for us in terms of dealing with them [Progress Software Corp.] is to have... they have coined a new term called 'federated databases'. We need to see that. We need to see better inline subroutine calls. We need to inline subroutine calls period to reduce the amount of code...

What shortcomings do you see in the prototyping method?

In cases where we don't think we can prototype, we turn down the work, so we lose work. I mean, it actually costs us money because we require that our clients work with us in the way we want to work. In fact, I have turned down incredible amounts of work...

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

[Not asked.]

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

The developer becomes the end user and then is treated as the end user. [Interviewer probes to see if experience can substitute:] The answer is yes, for a good portion of the development work, he [the experienced developer] can in fact, if his relationship with the client is such that they trust that he knows what he's doing... [Further probing:] He could probably get 75%, maybe 80% of the work...

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

Size of vertical market and the willingness of the funding source to act as as beta test and to work with us during the prototype phase. We turned out marginally useful code initially real quick [sic], and then they just helped us refine it.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

We feel very good. We don't feel the..., some of my people feel the anxiety, but I don't feel the anxiety at all, of going into new projects. I love going into new projects.

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

[Not transcribed.]

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

[Not transcribed.]

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

[Transcribed toward beginning of interview:] Well, don't forget there was no reason for prototyping up until the mid-70's, perhaps, because there was... First of all, computer time was expensive. Second of all, everything was batch [processing], in fact, everything was batch up until the early 80's if you really look at it. How can you prototype in situations like that? And thirdly, there were no intelligent users of computer systems. Everybody was a non-intelligent user, which is very rote, clerical. So, how can you prototype? [In response to request for additional commentary:] There is hidden rigidity [in prototyping] that you're probably not aware of. There's more rigidity than meets the eye. Prototyping design requires a database design, and it's database design that drives the process, not the prototyping... [Detailed example given]... There is a certain amount of prototyping rigidity that consultants don't understand because they don't do the real work. And, that is the true basis for our prototyping... It's not diagramming as we used to know it, it's prototyping as we now know it... There is a methodology that no one is aware of... but there is an inherent methodology in it.

EXHIBIT P

Transcript of Responses to Selected Interview Questions

Respondent 14

Would you now share with me your own operating definition of prototyping?

I find a lot in what you described [refers to interviewer's introductory definition]. Customers are traditionally people that are not MIS professionals. So, none of the traditional buzzwords or terminology of the computer industry helps us. Most cannot also diagram easily what things will be like. So, I deliver as soon as possible a prototype that allows the end user, the customer, to, number one, feel that I have an idea of what they really want. Number two, it's real work, I consider... I don't throw away prototypes because of the tool that I use... I deliver... I show it to them, I don't actually deliver, it doesn't actually go in hand. I expect there to be plenty of... it's used as a communications tool, as the process starts, so that's why I like it. I'm not coming up with a good definition, I suppose, but in that we can find one.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

Interactive versus batch. The user interface was important because the operator may not be the one who helped design it. And the application itself is considered very unusual. So, we're not mimicking a manual system.

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

The only one I think of that really didn't go well was... the president of the organization wanted it done, but the people who would actually run it did not want it. So user-involvement was a detriment to being successful

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

Only slightly less... You must coordinate additional visits. It also means that a lot of the creativity that goes into computer systems happens along the way. The positive of it is that it lets me run two or three projects at one time,

but the calendar difference means it takes longer, in fact for me it takes longer to deliver something, because it might take me... But, that's because I have several projects going at once. But it's not because of the prototyping, it's a matter of doing business.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

Yes, because as a business I apply a surcharge to systems that need to be spec'd [have specifications written] before they're started, because I'm taking a lot of risk at that time. And, I expect the customer to change their mind, because, just from experience... They [customers] will change their mind and to minimize friction along the way I put in a fudge factor. So, in effect, it costs them more to not pay attention to the fact that when I tell them ahead of time that they're better off paying as they go with close scrutiny of what I'm doing. Prototyping lets them see what I'm doing along the way, so that they don't have to be afraid of it running away from them.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

[Not asked.]

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

I could see some fear, but you must have more focus than an analyst. You must have better people skills during prototyping, because there's a tendency for an end user to see some work done and be afraid to tell you they don't like. It's easier to dislike something that's intangible, at the moment it's a diagram, it's something on paper, than 'Here's what I brought, is it what you wanted?' type of attitude. So I think you can be more of a consultant and be able to say, set it up such that they [clients] know that if they don't like it, then that's fine, that part of the exercise is to find out the 'nots' as well as the 'likes'.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

When I was in the sales business, we used to construct dummies that looked like the system because the end user, not being MIS, needed something to feel like we understood. So

those early systems would usually not have real calculations in them. And, in fact, I would often bring the printouts of a terminal session with me, because sometimes the system would not actually be running or actually, in the 70's, the end user would be fascinated with the terminal and not what it was printing. So would actually just bring paper and show them, and in effect it was a prototype. And it also kept them from, of course, entering their input and expecting to get their correct output. Now, with the new tool that I have, I can put a lot of the computations in the system as I design it, and in fact computations are an English-like language so that the end user can often verify the computations as I go along.

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

We now have a tool in which we build a prototype and don't have to throw it away and we know it's not as efficient as a recoded one, especially in a computer language. However, at any time the user is likely to ask for changes, and if it was in a more efficient system all of a sudden the very efficiency would hurt any changes, ability to make changes... [If necessary for executional efficiency] there are layers of techniques we can apply to make it a little more efficient, but at the present time with microcomputers we use a hardware solution - fast processors.

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

We've met with the new owners [of prototyping tool, Revelation] a number of times. They're in Boston [close to respondent's location] and New York. We're small enough of a team to switch to a new product if we need to, but it would be painful.

What would it take to cause you to leave your prototyping tool and move on to another?

I think it's... you get very spoiled by a development tool. Its quirks, you get comfortable with the quirks, so you don't mind... It would be hard for me to find another environment in which the data is stored in variable length and each field can actually store a number of occurrences... We would probably move to Pick [Operating System], a true multi-user operating system available, of course, on micros and minis rather than try another micro product. [Interviewer probes as to what features would cause respondent to leave his prototyping tool:] A question like that means you have to talk about business, the ability to write good programming is only a portion of what we do, we run a business. We

like the ability that we are knowledge engineers, so to speak, of this. We are teachers, we have an audience for insights in the package. So, one of the things [that would cause respondent to leave his prototyping tool] is, it would be something new, something that allowed us to build applications quickly, hopefully variable length. And, the company coming in would have to have a very strong commitment to supporting small development companies like us. That's one of the biggest... I would say that the Cosmos company that makes Revelation [the respondent's prototyping tool] has not been able to support its developers very well, partly because of their size.

What features would you desire in a prototyping tool that are not currently available?

In this particular package [Revelation], I need a better non-procedural report writer, a much better one...

What shortcomings do you see in the prototyping method?

Of course there's always... if you go the wrong direction, there is the obvious loss to time it took to get to that direction, and to discard it, weighed with... But at least it's a visual representation of what a customer would get, I don't feel that the traditional specs [specifications], they [the specifications] still allow you to go in the wrong direction, and you don't find out 'till you're almost done that something's wrong. But, the biggest problem is the perceived loss [of time]. Number two, it takes more experience, I think. You must be very facile in the language to be able to prototype. And you can prototype with this tool [Revelation] in front of the customer.

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

[Not asked.]

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

Well, there's nothing like showing, being able to actually show another system with a lot of similarity. So, if something is already built that is similar, that can be used very efficiently, effectively in lieu of prototyping. I would do that myself... [Interviewer probes for areas in which prior experience cannot substitute for prototyping:] Where the application is unusual versus unique. Almost every application is unique, but being unusual means combining aspects of modules of an application in ways that don't

seem to be done in that kind of department, or it's just that that department does something unlike anything else in the developer's background or in general business.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

[I chose to prototype this project] because that's the way I do things. Besides which, it is a project in which there are not specs [specifications]. There's an existing system that they are very unhappy with.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

I feel I'm getting good at this. You're right, there is anxiety all the time. If it's fixed price, the anxiety is higher, because I'm taking on all the risk of not understanding what they really want. So, if I had a traditional approach, they would be paying for my specs [specifications], they would sign off on the specs and then my anxiety about going the wrong direction would be a profession feeling that 'Gee, we didn't get to it; now we have to re-negotiate' and restart from some point, but not as much as I have to bear the entire risk.

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

[Not transcribed.]

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

It's all over the map... [When clients desire changes] I don't get the feeling of anxiety or finger-pointing or anything. I was able to convince them that their money was well spent even though it wasn't right, because it is something that is so intangible that they've learned something in it all.

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

[Interviewer queries a 90-day warranty on a sample proposal that respondent has provided:] I think so [that prototyping permits this long a warranty to be provided]. Warranties go

into how you run a business and the psychology of everything. It's partly just... I do expect them [clients] to have some changes. I don't warranty performance, I warranty bugs, so to speak, implementation [errors]. This is a fixed price [contract] so I don't want to deliver something, and a week later they find something's wrong, and I don't want to turn around and tell them 'Oh, that's a bug, but it will cost you to get it fixed'. I don't like wielding that weapon on a customer, especially a place where I will have repeat business. So I traditionally offer at least a 30-day warranty. For comfort purposes, this one became a 90[-day warranty]. But I don't expect it to be exercised much at all. I do expect some, because anything that is complicated is going to have errors in it, little errors. And I guess I wouldn't put it heavily on the fact that I am prototyping. We won't be off track by the end.

EXHIBIT Q

Transcript of Responses to Selected Interview Questions

Respondent 15

Would you now share with me your own operating definition of prototyping?

You describe it very well [refers to interviewer's introductory definition]. I know about prototyping from Electrical Engineering, which the only one that I know of to build something that works. I felt your definition was very good in that respect. [Goes on to give a detailed example.]

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

I think the main common factor is systems that require an extensive amount of user input. Prototyping probably would not be too terribly successful for device drivers or highly internal sections of code. But, it's that user interface that's so important and that where 90% of the input comes from, [it] deals with user interface. It may go further [unintelligible]...

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

I don't think I could answer that [in that respondent has had no unsuccessful prototyping experiences]. I could guess, but it wouldn't be very [unintelligible]...

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

I think the key element... that I as the developer don't have to learn the user's job. I developed some other things [systems] without prototyping, and I've spent almost as much time learning what the user was doing as I did developing the system. And, all that learning time is up front, before anything comes out of development.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

Again, back to the same thing [refer to response to time question]. If you spend less time learning the user's job, again, to me that's not productive time, but it's time that costs you and sometimes it's quite expensive.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

Again, it comes down to the user interface. The main complaint of the second system that was developed [refers to a non-prototyped system] is that the user interface is so complex that the only person who's really going to be able to use it is the guy who developed it. Whereas in my system that I developed [refers to a comparable system that was prototyped] the user interface was the result of a lot of feedback from the users. It's more attuned to what the users are looking for.

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

I think prototyping really gets them [user needs] out, but it's not, it doesn't happen on the first iteration of the program. Sometimes the needs of the user don't come out until well into the project. In that respect, it [development] requires good design techniques in order to be able to incorporate those into your program... You have to have been able to have written a program that's flexible enough to handle those things [changes in user requests].

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

I'm very conservative, I go right down the middle of the road... If everything is dummied out [i.e., a simulation approach], you end up with a dummy program, that really doesn't tell you as much as you need. But, there are places where you can't design it until you've gotten a little bit of feedback from the users, of what they're looking for. [Interviewer probes: does respondent mean stubbing?:] Right [as opposed to pure simulation].

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

If I went back and coded this [prototyped product] for total efficiency, I might be able to gain... 10% [performance improvement], but not much more. The program is very modular, it's written in C, it's very modular. The largest routine I've got is maybe eight or ten pages of source code, with over 300 separate, distinct modules. Admittedly, I could go in and rewrite it, but at any point where I see [that] something's inefficient, it's usually... a single module... maybe three to five pages of source [code], and it's relatively easy to clean it up at any point in the development. There are between 5% and 10% of the routines which I have done that to, after we've nailed down exactly what we've got and we're happy with it. I'll go back to it and just simply sit down and go line by line and say 'Did I make mistakes? Did I do this the best way possible?'. Clean it up and make it faster. So in my opinion... it's easier to go back and clean up the weak areas rather than just starting from scratch.

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

IBM is the only company I know of that I would have to say is stable. DEC might be stable. No one else is... So, I tend not to depend on any one company or any company's tools. In that respect, if the company offers source code with their product, and I'm heavily involved with it, I'll buy the source code, just on the presumption that a year-and-a-half from now I'll find out a horrendous bug has showed up at the worst possible time that, even if they're [the tool-supplying company] are gone, there's hope to fix it. I try not to depend on any one tool or any one company, but I'd be very surprised if IBM disappeared.

What would it take to cause you to leave your prototyping tool and move on to another?

[Not transcribed.]

What features would you desire in a prototyping tool that are not currently available?

It's currently available, but it's lacking, and that's documentation. Documentation has been a sore spot in the computer industry since approximately 1940. The biggest problem I face is that I know there's an answer but finding that answer sometimes takes me an awfully long time... Documentation, I think, is the single thing that I'd like to see improved.

What shortcomings do you see in the prototyping method?

The biggest shortcoming is the elimination of the personal factor, and that is that you're dealing with a user community... You've got to filter out what this guy [user] personally wants versus what the user community wants as a whole. His personal desires are usually not the personal desires of everyone else... When a person comes to me and says they want something, I usually task that person with selling it to everyone else in the user community... I'd say that half the time they do that and the other half the time they realize that here was just something they wanted that was nice for them but not for anyone else...

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

...The technique of prototyping has allowed us to actually produce a marketable product far sooner than we would have before. And, in marketing that product, we've generated an income which has paid for the development...

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

I think that the degree that that experience would substitute is that he [the developer] can make himself another user. And I don't think that his input should have any more weight than any other given user of the system...

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

I wasn't happy with what had been done in the past [refers to previous version of system], specifically in this [respondent's client] company. This company has been noted to produce products where the result of the product was outstanding, best in the industry... What we haven't been noted for, and have been faulted severely, is user interface and functionality. And I wanted to try and correct that problem of building a user interface that users were happy with, not something that they tolerated yet detested because it was just so hard to use.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

I tend toward the less anxious. Again, I feel I've gotten more control over the project because I don't have predefined, inflexible set of rules to develop by. Nothing's worse than you have to do a project and find out that original definition was wrong, but it's cast in stone and you can't get it fixed.

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

Let's not use [the word] 'leadership', let's use 'coordinating'. I do tend to coordinate. We have two groups of clients, we have in-house clients and we have external... [External clients] give us less feedback that our in-house [clients], which consists of our sales consultants and our project staff.

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

Our users are fairly aggressive and generally are not hesitant to come up with new ideas. I come up with ideas myself and I'll take the user role there and go to one or two of them and I'll say 'How can such-and-such be of any value to you?'. Fortunately for me, most of the time they say 'Yes, that would be a good idea'; every once in a while, they'll say 'No, that would be worthless, why do that?'. [Interviewer probes for balance between developer and client presenting systems options:] It is, at this point it's well balanced and [unintelligible]...

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

[Interviewer probes on relationship of prototyping and financial operations of the consulting business:] ...I could see that, yes, that would be a very valid point. I'll keep that in mind when I do another project somewhere else. Because there certainly is, you're right, there certainly is a very visible flag waving that says 'This guy's [developer] done something for you'... [Interviewer invites other comments:] As I mentioned before, I liked what happened, I liked the way it went [refers to prototyping project]. Certainly, I would not hesitate in the slightest to do it [prototyping] again. I've found in the past, when developers were isolated from the problem, which is the user's really, they don't produce what the user really wanted. It's virtually impossible for that user to communicate to a programming staff or a technical staff what he really wants.

He can describe it but often times that programming staff doesn't really understand it, but will politely nod their head 'yes' and go off and do their own thing. [This results] in a program that's usable by the programmer but not usuable by anyone else... [When I work with users] the users are really developing this program, all I'm doing it taking their thoughts and putting them into something that controls the computer...

EXHIBIT R

Transcript of Responses to Selected Interview Questions

Respondent 16

Would you now share with me your own operating definition of prototyping?

We've got a project going now that I would use as the best definition I've got; we don't always do it that way. The programmer brought in the opening menues to the system. He had not done an extremely detailed analysis of the system, but we had interviewed them on a number of occasions, and the programmer had been there. And, he had met with them on his own a few times and gotten to them on the phone a few times and had a pretty good idea of what they wanted. And, he brought in for them the first three or four screens and showed a few of the lookup capacity... and we got a lot of feedback and spent about an hour-and-a-half showing it to the users and got an enthusiastic response from them and also some requests for changes and he felt at that point secure in finishing the project, which is what he's doing now.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

Reduction in training time. Not invariable, but I would say a reduction in programming time. And, I think it's the only way that the user can get the little whistle and bell that they're looking for... So, I would say a higher degree of user-friendliness, letting the user define that friendliness, which is the valid definition of it, there is not objective [standards of user-friendliness].

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

We had a major disaster that we had to eat a lot of expenses on. It had to do with the client being a real hard nose guy, the first programmer being a disaster, the second programmer moving back to California, the third programmer being a disaster, the fourth programmer being good but expensive. Now, I wouldn't fault prototyping, but the client brought up major problems long after he had had the program in use, long after. And, insisted that we fix it for free, months after. So, you could say that he had all the opportunity, early so say [that he didn't like the system], and he didn't. So, it's not a foolproof method by any means...

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

If you mean successfully delivered, I would say yes. [Interviewer probes:] Because clients don't know what they want, and they won't know what they want until they've had a chance to see it. Major issues came forward in that hour-and-a-half that we sat with the users [refers to specific prototyping experience previously discussed], major issues that they hadn't brought up before, and probably could not have brought up. They just didn't think of it. So I would say yes. If you have a demanding client, definitely. If you have a client who will receive whatever we give them, that's a luxury we don't often have.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

I think it [prototyping] can reduce overall cost, I'm just trying to figure out if it will reduce our cost. It can, it's not invariable. I would say that there is that possibility... [Interviewer probes:] The majority of clients seem to win their arguments with us when they ask us to take what seems to be finished product back in the shop and spruce it up... And I'm thinking of one program now that was delivered, paid for and which we hope to resell, but which the client really hasn't been using, they paid well over $2000 for a database, a particular kind of database, they haven't used it. And, the programmer is in California now. And he said, yes, he would do some changes for us, but it's very iffy, and we don't know. He can send disks back and if it's no good then... If all that [proper needs analysis] had been done at the time, if we had forced the client... if that had happened, it all would have been taken care of up-front. And now, it's getting a little dusty, and we've got to shake the dust off, and we've got re-educate ourself on how it works, and the programmer has to remember how it did the code, and the client has to remember, 'Why did I ask for this?'... It just gets old. And that's one specific instance that comes to mind.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

[Not asked.]

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

I don't think it's [prototyping] a substitute for flowcharting, we flowchart... What tends to happen is we get a meeting [with clients], and then we call them [clients] on the phone twice. And then we produce specs [specifications]; usually, we haven't been paid for them [the specifications]. We're trying to be careful, but we're not going to stay up till midnight on a job we don't have yet. We try to make it [the specifications] look as good as we can. I think that that's just the beginning, I don't know how you would get around that. We're not sending in a prototype version that is just the initial menu. So, it's got to be more than that, so I don't know how you get away from the flowchart.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

I think it should [include computations in prototype]. I think it largely gets by without the computations, but that's the sophistication of the user. See, a lot of our users are small businesses and they're not sophisticated. If you're talking [to sophisticated users]... I would expect you'd have to include the calculations... We have [done simulation prototypes].

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

[Respondent claims he is not qualified to answer question.]

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

Word of mouth. A lot of it is just a feeling in the gut, you call their 800-number a few times and see if you can get them to talk to you. Quality of promotional literature...

What would it take to cause you to leave your prototyping tool and move on to another?

We got somebody in here persuasive who was working for us. I would drop R:base [current prototyping tool] like a hot potato.

What features would you desire in a prototyping tool that are not currently available?

[Indicated need for a simulation tool; will be examining Dan Bricklin's Demo Program.]

What shortcomings do you see in the prototyping method?

If you're [the developer] not really going to listen then you're not going to hear what the client says, and it's not going to help. It would be real easy to bully them, and throw jargon at them, [saying] 'This is how it has to be'. As opposed to the kind who could say 'This is completely a plastic environment and it can go anywhere and I can throw this is in the trash can right now if that's what you say'... I don't know if it's the [prototyping] approach itself, but as with any tool, it it's not used right, it can just confirm what you already want to do...

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

[Response in a different section of interview:] See, we're on a flat rate [for the particular project being discussed]. All this is [irrelevant] if they're [clients] going to pay you [developer] for your hours every time you turn on your machine, but it wasn't, we had to bid it on a flat basis. We told them hours, but once you say 100 hours for this and 50 hours for that, then they say 'That 150 hours and that's how much we are paying you'. Even if takes us [developers] four years, that's how much... they use it as a bargaining tool. So, I think it [prototyping] helped us meet the time we had budgeted to us... If we're throwing in the satisfied customer at the end, I would say without the prototype it might have been different story. It definitely could have more [cost] without the prototype, it definitely could have. Because these people weren't going to accept just anything: it was going to work the way they wanted it to, or they weren't going to be happy, and I'm sure they would have sent us back and said 'Look, we're paying you good money, you have to fix 9/10ths of what we're asking you to for free'. In which case, we would have eaten that cost.

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

Not at all, zero. [Interviewer probes:] Each job is really distinctive. To say you could substitute it [previous experience for prototyping], it's just a one way channel, that's handing down technology to the masses, and it doesn't work that way.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

The client, the main user, is on her first job, fresh out of school, and she doesn't know... what she wants, but she is very excited about getting it.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

The anxiety comes after the prototype, because you're never sure you got it... It's a field prone to anxiety, very prone.

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

I would say less [leadership is required]. I thought that was the idea, to let the user take more of a leadership role....

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

I think it's healthy, because it's trend toward group participation and a little more democracy in what has been an ivory tower sort-of activity. And ivory towers usually deserve to be pulled down, and certainly this one does.

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

I think it's nice that academia is getting a little bit closer to the reality of the situation. So far, academia's typical role in terms of spinning off products and ideas that the rest of the world will take and try to make money on. That's fine, I think that's the main job of a university in any situation. But I think it's also useful to take a look a little bit closer at how the industry, how the business is pursuing [its objectives]...

EXHIBIT S

Transcript of Responses to Selected Interview Questions

Respondent 17

Would you now share with me your own operating definition of prototyping?

I liked your definition [refers to interviewer's introductory definition]. Basically, the work that I have been doing in Progress [respondent's prototyping tool], which is the work I've been doing in prototyping, always I sell it as 'OK, we'll start off with something small that will meet your most immediate need; don't expect it to be the final version. We will deliver it to you for X amount of dollars in so many weeks, and I want you to start using it and tell me what you think'. You see, I use it as sort-of a selling point. If you're [the client] happy with it, well then we can go on to step two, we can decide what your next immediate need is, and while we're refining the first one, we'll blend the second one in, and keep going until they say 'Enough', until they're fully happy or decide that they've paid as much money as they can afford, or whatever.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

I would consider them all [his prototyped projects] successful and that's because the client was totally involved the whole way, from the beginning to the end. I basically did what the client wanted and they were reviewing on a regular basis, so they never really got too far out of what they wanted. As soon as I began to stray from what they wanted, we talked about it and decided on a course of action. So, I would say client involvement is the most important aspect. [See also comments in response to time question.]

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

I don't really have that much experience with things that were unsuccessful.

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

Yes, but, the reason that it's 'yes, but' is that you're delivering something from day one that everyone understands is a prototype. If you're talking about final delivery, when you stop work, I would say calendar time, it's much faster doing the traditional design and build... The fact is that prototyping is an iterative process, and those iterations have a lot of redundancy built into them, which is, I think, a positive thing because it increases client [involvement]... from my point of view, my measurement of success is whether the client is using, whether the client is happy. And, to me, the way you do that is involve them, if you want a successful project you have to involve them. You don't have to, but it sure helps to involve them every step of the way in order to give them a feeling of ownership, giving them a feeling [of] investment, and once they feel that they own it and they've invested in it, and money won't do it. You can ask a client for money and he still won't have a feeling of investment... to me, it's the time and the sweat and the iterative process that's so necessary for a successful project.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

It did occur to me that there is a difference here, and the work I have done in prototyping has always been for one client and the idea was that... we would eventually turn this into a product that we'll market, an off-the-shelf package. But we really haven't done that yet, we've just been too busy doing one client at a time, and prototyping for that one client. And that makes a big difference, because the kind of thing that you design and build [using conventional development methods], you probably design it with the idea of many clients, at least you should..., and then you sell it many times. And so when you get to costs, I would say that the cost is decreased if you're looking at the one client, because you're giving that client exactly what they want, at least I do, with very little regard for the generality and therefore you don't get into a lot of issues where you're making your job difficult because you're trying to build in generalities. My feeling is, the client is paying for it, you give him exactly what he wants, and don't spend your time building in generalities. Do that on your own time later. And, I have never gotten to that stage yet.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

I've got two answers. One is on an individual basis, prototyping something for one client, he's going to get better quality because it's going to be exactly what he wants and needs. If you are developing something for off-the-shelf

packaging... you're going to get a better product by sitting down at the beginning, doing your homework, figuring out exactly what everything should do, and design it properly and do it right. And I think you're going to end up with a better quality product overall that way. It's good for a mass market type product.

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

Prototyping, because you are getting a lot more client interaction, in my opinion... I think that that doesn't have to be so. Obviously, if you're going to design and build [conventional development methods], which to me is the opposite [of prototyping]..., there is no question that you interview a lot of clients, get a lot of client interaction, as much as you do with prototyping, but the difference is that the client doesn't necessarily talk your language when you're sitting there up-front waving your hands and describing something. You have a picture of what you're talking about and they [clients] could have a very different picture, but if you have a prototype where you can sit there with him [the client] and show him what you've done and he can play with it, now he sees what you're talking about and you hear what he's talking about much better with the actual thing there and being played with. And that's why I consider the user-involvement of a much higher quality and what not.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

[Favors including computations in prototype.]

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

I disagree with that [recoding for executional efficiency] for two reasons. One is with modern tools, that's not nearly as true as it used to be. And, number two, even if it is less efficient than it could be if it were programmed in a lower level language, in this day and age, that's stupid, that the cost of people relative to the cost of the MIPS [millions of instructions per second] is way out of whack... now, MIPS are cheap and people are expensive, and it's not worth your while. If it's sluggish, they buy a bigger computer, but don't spend millions on rewriting the code into

some lower level language that's not easily maintained and what not.

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

...The reasons why I judge it is one of the founders [of the firm producing respondent's prototyping tool] was my roommate and I know them. They have two senior vice presidents, and they were both roommates of my mine as some point, we went to school together, college buddies and I've got very high regard for both of them. I've also done some consulting for them and gotten to know quite a few people inside [the firm] and have been impressed with a lot of people I've met. So, basically, I've been not only with the product but also the people, the key people in the company.

What would it take to cause you to leave your prototyping tool and move on to another?

Three things. One is if a client really wants it in another language [primarily refers to tools, not third generation languages], we'll put it in whatever language the client wants. And, in our case that very well could happen. The second thing is money, that [moving to a multi-user version of prototyping tool, Progress, could be tool expensive for the consulting firm]... The third reasons is that, in my business, we deal primarily with Fortune-500 companies and most of our clients have large [DEC] VAXs, large IBM mainframes and several of them have Crays [supercomputers]. And, Progress [respondent's prototyping tool] will run on the VAXs [but not on other mainframes]...

What features would you desire in a prototyping tool that are not currently available?

There is not much that Progress [respondent's prototyping tool] doesn't have that I would desire. In fact, there are some features that if I were rational I would desire, but I'm not rational about it. For example, they [Progress Software Corporation] are developing an applications generator system, which is actually pretty spiffy. The one thing I would like to see them do that they aren't currently planning... it [the applications generator] is taking all the challenge out of programming, and that saddened me. And I've read... that a lot of programmers had the same reaction, even though they admit that it will generate better, cleaner code and bug-free because all the modules are already there and you just put it together, people like me still like to program. ...I don't really desire it, even though I appreciate it, I think it's probably good and I should use it anyway despite the fact that I really don't like it. Now, the other thing that I really need, and this is not so much having anything to do with prototyping, is just has to do with the business that I'm in, a lot of our

clients really want a true distributed database system, and we're talking about a true one, everyone's advertising them now but nobody's got one... and that would help us immensely, but that's got nothing to do with prototyping.

What shortcomings do you see in the prototyping method?

The major shortcoming I see is, and I don't know whether this is prototyping or this is Progress [respondent's prototyping tool], but I suspect it's both, I suspect it's prototyping and it's exacerbated by Progress, and that is... Progress does an awful lot for you, and as a result when you put in a few lines of code, it does incredible amount of things for you. And, if you're totally on top of it, it's what you want, but because it's doing so much for you that you forget that by changing... by adding [a few] lines of code, you've now changed the scope of a transaction, or you've changed the scope of a record, or something like that. And, once you do that, all hell can break loose, so there's a lot of side effects, is what I'm trying to say... So, to me the problem is that you develop some system that works and tried it out and everything seems fine, but there's a lot of hidden things because you never tried out. And, there are lot of side effects, particularly in Progress, but I think in most modern languages have that. That's where modern 4GLs [fourth generation languages] are going to: where one simple statement does an awful lot for you. And, trying to control all aspects of it. Very few people in this world, I think, know how to control all the aspects of Progress...

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

Prototyping is how we do our business. We have a product in mind that we expect to make us all very wealthy. One way of developing that, because it's going to be two or three years in development, is to go out and get venture capital and give away a large part of your company or what have you. But, instead we chose the route of prototyping for each individual client to develop this big product, we'll put all the pieces together to develop this big product. Without the prototyping methodology, we couldn't do that.

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

In my own experience, I've known something cold, absolutely cold, it's much better for me to just sit down and write the thing and not worry about the prototype because I know all the issues that are going to come up and I know how I'm

going to solve them. And so that is better, I would say, than prototyping. I would say that you get a better product faster, cheaper and everything if you have someone who's good, who's really experienced in that area...

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

[I choose to prototype this project] because this one... is putting together all kinds of data that no one put together before. We really didn't know how to proceed but we decided we'd start with the most important data, it was the data we needed for our simulation model [to be used by modeling software, not for a simulation approach to prototyping] and then just go from there and just expand and expand and expand, all the time keeping extremely user-friendly and extremely professional, just the appearance of it.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

[Not transcribed.]

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

I think it's the same both ways. I mean, you have to have good communication. I don't really see it... In both cases [dealing with anxiety and leadership questions] I am ultimately responsible for the success of the thing [system developed]. I always feel that I have to be responsible but that also means I must responsibly listen to what their [client's] desires are. I just can't just go off on my own, so if that's leadership then it's the same.

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

I've found that it's pretty much 50-50... I find that most of my suggestions are gladly taken, and I also find that most of their [client's] suggestions, I'm more than happy to do for them because it makes good sense to me.

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

I think [financial considerations] is one of the key things to prototyping... I firmly believe that what makes me personally happy is a happy client and that's foremost. The second thing, is to have the client using what you have developed. I'd be very saddened if I found, even if they liked it, it never got used. And, I've heard so many stories... it hasn't happened to me personally... where they developed a nice system and the client thought it was a nice system but they [the client] never used it for one reason or another. And I find that that doesn't happen with this iterative prototyping cycle, that it gets the client very much much involved, it develops some very strong personal bonds. It helps your business even beyond that because now, what we've found in these big companies, is that our original client is so happy with our work, because he's worked with us and got to know us. Quite frankly, in our company we don't have the typical salesman type of person that comes in, very well dressed, looks like he was an all-star football player and glad-hands and everybody swoons..., we don't have people like that. We have very smart people that don't necessarily have the great social graces. And so, I think it's difficult for us to get the initial foot in the door because we don't have that great first image. But, what is incredible to me is all the follow-up business we end up getting, because once the people [clients] start working with us... See, that's the other thing, these football players are out there in the forefront, they don't actually do the work, and even though the people like the guy who made the sale, the guy actually doing the work they may not like, but they actually work with us, they like us, we get to be personal friends, and what we've found is people within the [client] company will start [to make referrals]... And I think this personal contact is so important for even marketing. The one thing that bothers me in my mind, and we've talked about this, was this issue of as you prototype, especially with the fourth generation languages that do an awful lot for you, it's very difficult to my mind to keep totally on top of things... I feel I've given a lot of that [sense of control respondent feels with third generation language development] up with the prototyping method, and that disturbs me, because my office might be a mess, but my mind is always crystal clear and with some of these things [prototyping tools] I tend to get foggy... And that doesn't happen to me with the [conventional] design and build.

EXHIBIT T

Transcript of Responses to Selected Interview Questions

Respondent 18

Would you now share with me your own operating definition of prototyping?

Prototyping is a way of getting a client to visualize the end product without having to develop the whole thing. It's a first step towards getting feedback from the end user and it's also meant to be plastic, so in other words, you do a prototype, you show it to the client to see... it's harder to see a system on paper than it is to see on the computer, so they see something on a computer and they notice things that they wouldn't notice before, and then you incorporate it. It's a feedback mechanism for users.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

Knowledgeable users, I think is the key. You have users who either understood the idea of feedback, of getting involved quickly, or they were educable.

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

It's where users aren't involved, where they [users] assume you turn a computer on and magically it does whatever they want... [This] is especially dangerous in the microcomputer area because I think some of the major systems, IBM, Apple... they kind of come up with this idea that computers are fun, easy and there's no work involved, and really there's a lot of work involved.

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

Less [calendar time]. You identify bottlenecks quickly, which means you can anticipate situations where you have to get other people involved, other revisions of the specs [specifications], that obviously makes things quicker.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

Less cost, there's no doubt about it. It's very simple, time is money, and the quicker than you can see or anticipate problems and pinpoint them the quicker they'll be identified and resolved and that means it costs less.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

Greater [quality]. Since the user sees what they're getting, then they're involved in the design, it's their system ultimately. I think in any successful systems implementation, you have to have the users... it's basically the users have to buy off on it [new system], and there's a qualitative difference if they just sign off on some piece of paper versus they know they participated in it from an early stage.

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

...there's a group of experts out there [conventional analysts and designers] that really know, sort of like the guru approach. I don't believe [in] that. I think that's a mentality that is sort-of old-fashioned. It's a mentality that I think most computer people had ten years ago, and maybe was true ten years ago. But, more and more users are getting more involved in systems design anyway, especially with the advent of microcomputers. And, I think users know what they want. The problem may be that they can't articulate exactly what they want in the language that the developer, the systems analysts or the programmers need to implement it, but that's changing also. There are tools that are coming out that are more user-oriented so they are articulate what they want right on the system. So, the idea that there is a priesthood out there that really knows what's going on... I know that's not true.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

I think you should some computations, the critical computations [in the system], up to a point. I think you need enough so that the user just doesn't see screens, they have to see some logic that's going on, it's got to be believable. There's a fine line between doing the whole system

[coding included] and giving them just the screens [simulation approach]. I think you've got to go beyond just the visual part, and show something that works. I would do a piece, a small piece, that you could sense is critical to them, and get that to work, rather than say 'I'll give you all the screens' and not have anything work. Also, it's important for the user to see the style of the data entry and the reports, and so on, to see if it meets their requirements

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

I never worry about performance when I develop, at least not at the beginning. I look at functionality and try to get the thing, try to get a prototype that mimics how the user is going to be using it. If performance issues are identified, then you address them locally. In most of the time, it's something that's manageable, it's not that you have to throw out everything just because there's a performance bottleneck somewhere, you manage it locally.

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

Cosmos [developers of Revelation, respondent's prototyping tool] has been around since 1983. They have a substantial user base, about thirty or forty thousand users, and the tool works. We don't really need that much support. Basically, we sell Revelation so I guess we have to have a source for it. As far as support, we're sort of self-supporting, in most things. We certainly hope they do well, but they don't have to do that well, they don't have to become another dBase [a popular microcomputer database management system] for us to survive.

What would it take to cause you to leave your prototyping tool and move on to another?

Superior functionality. One other thing, the market has changed so much that maybe everybody went to another type of machine or another type of environment, which Revelation [respondent's prototyping tool] didn't support. It might be if the company [Cosmos, developer of Revelation] certainly would account for that. But mostly, it's whether it solves the problem, and as long as we are encountering situations where Revelation will work, I think we'll keep. If it doesn't, [unintelligible]. We're kind of customer-driven more than anything else.

What features would you desire in a prototyping tool that are not currently available?

What would be nice is a little better... you have to write some utilities to organize all the data items in all the dictionaries, or all the screens, it's not centrally managed, they [Revelation, respondent's prototyping tool] don't have like a central management facility for data items or... You have to impose a structure on what fields you define, what files you define ...it's kind of ironic to say, I use this prototyping tool and it doesn't organize the data, but not globally, application-wide it doesn't. It would be a nice thing for it to do.

What shortcomings do you see in the prototyping method?

You could go into a seat-of-the-pants mode, if you sort-of misuse it [prototyping]. And, there is a tendency to sit down and start getting something out and not thinking about the whole picture. You still have to thing about where the whole picture is, and you can't just say well, I'm not going to worry about the whole environment of the application. It's something that you always worry about, who's going to use it? You do things a little differently if an expert, a database expert's going to use it, someone moderately expert versus a naive user, and so on.

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

It [prototyping] helps [business operations]. You get a clearer picture of what the project's going to be. And we find we can bill people for the prototype, because it's a value added to them. It actually gets us off the idea... if we tell them [clients] it's an interactive, they participate in it type of deal, then they feel that they're getting value, and therefore it's not like we have to build you something first and then discuss what the ultimate cost is going to be, because you can kind of bill them hourly. So, it actually helps a lot.

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

That's very good, if you have a developer who understands the environment, the way the application is going to be used, they've effectively prototyped it before they started. [Interviewer probes regarding where previous experience would not be helpful:] They might get too close to the thing. They might think that their way... is the only way, it has to be done that way. And, they don't think of all

the alternative ways of doing it. You can get kind-of fixed, tunnel vision.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

Because I knew the end user, the end user is actually doing most of the prototyping, and I knew that he was knowledgeable enough to do it. And, basically I come in to fill out the details, so it works very nicely. The end user gets involved, and he likes it...

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

I have no anxiety at all. My feeling is, eventually I'll find out what they [clients] want. I don't expect to get all the answers the first time, in fact... I try to get as much as I can, but I never get it all the first time. And that's what nice about prototyping. I say 'Look, you've given me a lot of information... I'll feedback to you [client] what I heard from you'. And, invariably I do that and find that I didn't quite get it all right and I keep going. It's an iterative kind of thing.

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

Less often. I don't believe I'm... I don't see myself... I see a partnership role, is really what it is.

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

There are times when they [clients] ask me for various options, and there are times when I have to defer to them. It's sort of half-and-half. As I said, I try not to make it a leader-follower kind of thing, maybe that goes back to that priesthood idea [see respondent's comments in needs question], I don't really believe in it. I think it's a partnership. I believe the customer knows what they want, that's the first thing, and that's counter to what a lot of traditional data processors feel, this idea of 'dumb end users' or whatever. I don't believe it, and I believe my job is to interpret what they want, and to assist them, and to feed it back to them, until we get it right.

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

I think prototyping is an excellent tool, I'm definitely an advocate of it. I think it's going to... more-and-more there are going to be tools that allow... get away from the pure programming. I haven't done pure programming in a long time, I write little subroutines now, but they're ways within the structure of Revelation [respondent's prototyping tool]. So, it's not like I start a system from scratch, and I think that's the way things are. I think there are going to be tools increasingly available, and that users will understand, it will be more at the user level. I think graphics is an area that people haven't really addressed as much, as an area where prototyping is going to help. To see a graph or a chart of the relationship between the entities in the database, you need some good hardware to do that, some really powerful graphics stuff, that's an area which I think is going to be nice, it's going to help in the prototyping process. And I think users are more-and-more sophisticated now. You can't get away with... you don't get a whole lot of people now who say 'What's a computer?, What's a disk drive?', that kind of thing they used to do five years ago. People have real, honest-to-goodness needs and requirements of systems. They ask very tough questions now and you got to have answers that will do it. There's definitely enough problems, enough things that have... problems crying out for solutions that prototyping can only help. That's about all I can say about it.

EXHIBIT U

Transcript of Responses to Selected Interview Questions

Respondent 19

Would you now share with me your own operating definition of prototyping?

To me, prototyping is developing a subset of the system, trying it out, seeing if the concept is workable, seeing if it's economically feasible.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

They [developers] have been given time to really work out what exactly was needed, or wanted, or desired. To work out bugs.

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

[Respondent could not think of answer.]

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

I don't think that [prototyped systems can be delivered in less calendar time] is necessarily true. There's probably less frustration, because the expectations up front are different and [unintelligible].

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

Probably. Because I think you spend less time reworking and changing... [unintelligible]. It's easier to change the code when there's less of it than when you're at the back end [of code development].

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

Well that I think that somebody who has developed a product or a system has a certain amount of pride and will work to

get to as good a point as possible, and he'll do that whether he's prototyping or not, it's just a matter of how long it's going to take him, what it's going to cost.

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

I think you need a combination of both [prototyping and conventional development techniques]. I don't think you can prototype unless you sit down and do some traditional systems analysis up front. And, traditional systems analysis requires interviewing the user, which you to do to prototype also.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

I think the users are less inclined to use it if it's not a real, live situation.

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

It depends on what you code the prototype [in] to begin with, I would think, and what you are ultimately going to code the program in. Not everybody codes in assembler, in fact most people don't code in assembler anymore, and everything is less efficient than assembler. [Interviewer probes:] I think if you start coding with that [evolutionary development] in mind, and you're a good programmer and you code the prototype efficiently...

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

[Question not relevant: no experience in prototyping.]

What would it take to cause you to leave your prototyping tool and move on to another?

[Question not relevant: no experience in prototyping.]

What features would you desire in a prototyping tool that are not currently available?

[Question not relevant: no experience in prototyping.]

What shortcomings do you see in the prototyping method?

[Question not relevant: no experience in prototyping.]

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

[Question not relevant: no experience in prototyping.]

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

[Question not relevant: no experience in prototyping.]

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

[Question not relevant: no experience in prototyping.]

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Question not relevant: no experience in prototyping.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

[Question not relevant: no experience in prototyping.]

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

[Question not relevant: no experience in prototyping.]

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

[Question not relevant: no experience in prototyping.]

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

[Interviewer probes earlier remark that prototyping is not appropriate when the developer is under time pressure:] I think it depend on what you're using as a prototyping tool, maybe. If I spend time with a picture show type thing [a simulation program], that takes away time from coding. If I spend time with an applications generator, and then have to

go back and recode, that's extra time. I would also like to think, which may or may not be true, that I plan fairly well and so when I'm under the gun [time pressure], it's better to have a good plan. I also am trying to be very strict with myself and not do a lot of embellishment up front, people get hooked on doing screen designs and that kind of thing, and I try not to do that until the very end, so that helps too... [unintelligible]...

EXHIBIT V

Transcript of Responses to Selected Interview Questions

Respondent 20

Would you now share with me your own operating definition of prototyping?

...It is a first draft of a working system, that would show them [clients] the inputs and the results of the system and would show us [developers] the critical issues of database structure and data flow and... we're not going to decide at this point whether this will be a prototype [implies modeling development] or the first draft of a final system that would be simply a polishing up of the first draft...

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

My experience isn't broad enough to know of a lot of those [systems that have been successfully prototyped].

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

[Respondent did not respond to question.]

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

I think that they [prototyped systems] would be better because there is more back-and-forth [interaction between client and developer]. There is a certain amount of inefficiency that comes from trying something, going back, the back-and-forth, I think, may actually consume some calendar time. I don't think that's a bad thing; I think in the end you probably end up with a much stronger system that will last longer, be good for a longer time.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

My suspicion is probably the cost is about the same [comparing prototyped and non-prototyped systems]. I think it's the quality and... the extent of user involvement, commit-

ment and ownership of the system that makes the real difference.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

[Not asked.]

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

A good systems developer doesn't talk once to the client and then go back to his office and design something and go ahead with it. There's some prototyping even without using a computer at all, that can take place merely... drafts of reports.... To set on one side columns of variables and variable definitions, and on the other side drafts of the reports that we output by the system, if you have somebody doing the development process who knows or can think through the process by which the inputs become the outputs, he can do a lot of prototyping in his head without sitting down and programming. And, if he talks to the user once, prepares an initial draft of the kind then goes back to the user and says 'This is how it will work, here is a sketch of the menues you will use, here are first drafts of the reports. Is this what you want?'. And, the user says 'No, this is what I want.' And, he [the developer] goes back and he does it again, that's prototyping of another sort that takes place without actually programming. And, if you do that well, it seems to me it's almost as good as using the machine and maybe it's more efficient because you don't have to consume time creating code. I don't know.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

I would trust the working version more, in the sense that I think what's really critical for system development, database system development, is the database structure itself, and the flow of data. Something that is simply a user menu that picks out word-processed reports that have nothing to do with data that's put in, that's just a cosmetic.

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

I think the technology is changing in a way that allows a different way of coding. It's true, that one of the arguments for doing all of your system design before you ever start programming is that then you can minimize storage and memory usage and speed up the process. And, I think that's still true but as machines get faster and faster, inefficiency matters less and less. That's true in this case... In a way, the faster, more efficient machines allow for more intuitive, wayward, exploratory kinds of development procedures...

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

I did a reasonably thorough check of Microrim [supplier of R:base, respondent's prototyping tool], not just the Dun & Bradstreet report but I also talked directly to the controller. And, he was very frank. He told me how much cash he had, how much... he gave me a sketch of his balance [sheet]. I talked to people who were smart about [Microrim]...

What would it take to cause you to leave your prototyping tool and move on to another?

Something better. Nothing that's available now.

What features would you desire in a prototyping tool that are not currently available?

It would be nice, for example, to... I found that when I was presenting things to clients that I often would fake part of it with a spreadsheet, and spreadsheets are nice as is word processing for reports. If R:base [respondent's prototyping tool] had an integrated spreadsheet as part of it, that is, where you can create a screen report, then you could insert columns in, and insert formulas into the columns, without treating it like a spreadsheet, without a full-edit screen as [in] a spreadsheet, then that would be very powerful.

What shortcomings do you see in the prototyping method?

Well, there is always the danger, it's like the habits you get into when composing on a word processor, it's the danger that since it's so easy to create a first draft, you get stuck with the clumsy expression you use in the first draft and pretty them up by moving them around, and that kind of writing. As somebody who has taught writing for years, I know that sometimes the best thing to do with a first draft is to shove it in a drawer and start all over again, and not

try to rearrange paragraphs, rearrange sentences, rewrite details, but clear your mind and start all over again. One of the dangers of prototyping may be that you're not forced to do that. There are tradeoffs on both sides.

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

[Not asked.]

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

Well, that is something that I've thought about... I'm sure that it does.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

Because of my relationship with the other consultant [on this project] and my own transitional career path. That is, I was moving into full-time consulting and learning R:base [respondent's prototyping tool]...

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

I guess less anxious. You can keep things working, solve problems as they come up. Sketch out something, and you're glad that it works. You may change it later. I think your client gets a little bit of confidence too, because they start see results earlier.

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

[Not answered.]

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

I don't know. It's a back-and-forth. I guess I encourage them to do that kind of thinking on their own, but I present options as well, so that it's a dialogue.

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

I guess one thing that characterizes me as a programmer and system developer is that I really regard this as play and as fun. I get involved in my programming as I would if I were doing [a hobby]... What attracts me about this kind of work is the problem solving, not the kind of grit-your-teeth-get-this-problem-solved way but the way an intellectual puzzle [is solved]... It seems to me [to be] intellectual play. That's what I like about it and that's one reason why the prototype method appeals to me. It's sort of creating toys that behave like real systems and then turning the toys into the real system. And that seems fun to me... I think it's easier to do prototyping, and moving from prototype to finished product, when you have one person who's centrally involved in both... I think there is some question about a team effort, and whether a team effort in which one person does the prototype and another person does the final, that might introduce some inefficiency. On the other hand, if you can have one person centrally responsible for both the prototype and the final version, I think it's a very powerful for that one person to develop his own understanding of the system...

EXHIBIT W

Transcript of Responses to Selected Interview Questions

Respondent 21

Would you now share with me your own operating definition of prototyping?

I agree with your definition [interviewer's introductory definition], I like that definition. That's a well-defined thing, what you described. I think there are other things that you can do that could be called prototyping, and maybe we should have different words for these different things. I'm sure this is what you are struggling with. By your definition of prototyping, I not sure that I've ever actually done prototyping, and the difference is that I don't think I've ever actually delivered something to the client, to the end user, and have them really give much feedback on it. The kind of prototyping I do, and I never really realized it till I listened carefully to your definition, is more of an internal process. I will go through a prototyping cycle for myself, and part of this is reflection of the fact that in most of the projects I've dealt with, I've had a lot of leeway in designing the user interface. I have not been... I've been part of very small development projects, with only a couple of people working on them. And, there hasn't been a formal requirements phase and a design phase, where it gets all written down on paper, exactly what keystrokes are going to do what. So I'm free to suggest whatever I want in that, and in many cases just do it, where the client says 'We trust you to develop it' or 'I'm [the developer] responsible for it'. And for my own... in some sense, I'm kind of the end user there, although not really the end user, and I will do my own internal prototyping... that's my style of development, to get something up and running as quickly as possible, to see what it feels like, both in terms of functionality, the user interface, what it looks like to the user and in terms of the software development, the internals. So I will use a rapid prototype not just for what it will be like for the user to use the product, but what will it be like for me the rest of it, to flesh out the details. So, I can be prototyping both the software interface and this internal structure of the program itself.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

I can't answer that because the prototyping that I've been exposed to has been so informal and not really a major conscious part of it at all, it's just kind of a nice way of doing things.

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

I can't think of any examples of that [where prototyping was attempted but did not succeed]. I haven't been exposed to that.

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

It's hard to say - it depends on what you call 'delivered'. In a traditional life cycle, I can imagine two points in time. I can imagine delivery and sometime later when it's really working. And, I would imagine that the delivery time of a prototype system would fall somewhere in between, that there might be a little extra calendar time involved in actually getting the thing initially installed, into production. But, it would be more likely to be really workable at that point in time, and not require lots of hassles until you get it in place... [This opinion] is just a complete gut feel based on my exposure to various environment. To be honest, I may even be partly influenced by claims of people who I read five years ago [in graduate school] who were advocating prototyping. That's kind of what it's supposed to do, I don't have any first hand experience. But, its consonant with my intuition about it...

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

I think the key is that it keeps people in the software development process... A couple of things:... I think, especially in larger software development projects... in my [smaller] projects, people tend to get bogged down because people who work with computer systems and software, programmers, analysts, whatever you want to call them, thrive on quick feedback. You put somebody on a project that's big, they're not going to see anything working for a year or a year-and-a-half, it's hard to be motivated. I think rapid prototyping improves motivation. I've seen, in my indirect exposure to larger projects... the level of enthusiasm and involvement in the project comes to a head when you're putting it together and seeing it on the screen, work. In a traditional methodology, that happens at the very end, in a relatively small period of time... and with rapid prototyping you're getting little dribs and drabs of that throughout the cycle, and I think maintains the excitement and the involvement and the goal-directedness of the people on the

project and I think that's why [overall costs are lower]. [Interviewer probes for relationship to cost issue:] Because I think the thing that makes software cost a lot of money is that you're paying is low productivity on the part of programmers, low productivity because of low motivation... and also because of dead-ends. People will have design meetings where they'll argue about whether this way is a better way to do it, or that way is a better way to do it, or how to document this at the high level, and all this kind of stuff, whereas if you're working towards a prototype of that piece [project], somehow my experience is, the closer the implementation is, to get your hands on something real, the easier it is to break through all the arguments and discussion and stuff that goes on and get down to an agreement and see how it's going to work.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

[Not asked.]

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

I think I'm getting caught between two different interpretations or uses of the prototyping technique, and one is internal and one is external. And you're [to interviewer] really focusing on the external, where the purpose of the prototyping is to get user feedback, make sure that the design meets the needs of the user. I think of it more in terms of ways of making the software development process more efficient, better motivated, more on target. I would use prototyping to prototype an internal data structure, to see how efficient it is for accessing the data, it would have nothing to do with the user. It terms of eliciting user needs, I'm not sure that that kind of use of prototyping, I don't have as much instinct about that use of prototyping. And, my gut feeling is, I'd be reluctant to replace traditional methods of eliciting user needs because I found that users often don't bring sufficient understanding of how computers work to their own understanding of their needs. They know what they need, but they don't know how a computer can best meet those needs. And, I think the traditional methods attempt to bridge that gap, of combining the user's understanding of what they do and the analyst's understanding of what a computer can do to automate that, and working that out. And, if you just... I think prototyping can be a useful tool to help that, but I don't think it should be an either-or type of thing. It's just my gut feel.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

I would like to see computations put in whenever it is feasible to put them in, partly because that makes it more useful for what I like to use prototypes for. And I think also it will help the user. I could imagine a life cycle phasing of prototypes during the early stages with no computations, just playing with screen layouts and things like that, and then... it would seem to me to be a waste to then say, 'OK, now we've got a user interface design, now we're going back and do the rest of it using a traditional methodology, and we're never going to back to our prototype again'. I'd like to see the prototype evolve with the life cycle along with everything else. In fact, I like to see a seamless evolution from prototype to finished product, rather than throwaway.

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

[Not asked.]

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

I want source code... The second thing is, I strongly prefer the ability to use it [prototyping environment] royalty-free. I have not philosophical objection to it, I just don't want to have to deal with the administrative problems of paying royalties... [Regarding C-Tree, respondent's prototyping environment] I'd heard word-of-mouth good things about the product. I'm less concerned about longevity if they give me source code... The other thing that I look is the the range of environments in which their tools are supported. That's a key thing. If they publish a list saying our tools have been used on [lists numerous computing environments], then I have a good feeling. Two reasons. One is I think they'll be able to withstand shifts in the marketplace... Also, if I ever have a client who wants me to do something in that environment, what I've developed around their [prototyping tool vendor's] core is usable.

What would it take to cause you to leave your prototyping tool and move on to another?

[Not asked.]

What features would you desire in a prototyping tool that are not currently available?

In the screen area, I would like more machine independence. My current environment is closely tied to the IBM PC -- machine and operating system independence. I would like it to run under OS/2, with dumb terminals, attached terminals... and so on. The other thing that I'm missing that I would like, whether this is prototyping or not, and again I believe in prototyping all aspects of the system, and prototyping and reusable software and so on are all closely intertwined. And, one of the things that I am missing is a general-purpose report generation and data query mechanism. If there could be standard way of doing that, and I could get a tool that would allow me to quickly prototype things like that...

What shortcomings do you see in the prototyping method?

I suppose there is a risk that you get too driven, too blindly driven by user requests. That's probably the most severe possible drawback that I can think of. In the sense that the user will look at the prototype and say... In my experience, I've been asked for things from the user where they think where they think of something that they do occasionally and that they can't do easily on the screen and they [request it]. And, when you think about it you discover that the effort that it would take to put that into the computer system is not worth it because this is something that they only do once in a while. But the user has just asked for it because they don't know what's hard and what's easy, what's worth doing and what isn't. And, if you get too driven by user feedback on a prototype you might not stop and say....

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

[Not asked.]

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

I'll tell you what I would like to see happen in that situation. I would like to see a group of users, user representatives, sit down with the system that the developer has familiarity with, and use it as a prototype. And, react to it. And, at least sign off and saying 'Yes, I agree, I like that and you [the developer] do something just like that for me and I'll be happy'.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

Because I felt that the user needed to be convinced of the feasibility of the project.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

Less anxious... you want fast feedback, that's what people whose brains work in that way [systems developers] thrive on, that's why people become hackers and get engrossed in computers, they want the rapid feedback. And I find it very anxiety-provoking to have to stop and walk away from the rapid feedback environment and sit down and write reports and papers and design documents and so on and so forth.

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

The answer is yes [must take on leadership role more often] but that's kind of a statistical artifact. It's because when I have control of the project, I do more prototyping kinds of things. And when I'm working as part of a larger project or part of a larger environment, when I have less control, I'm more likely to be dictated to by traditional methodologies. So, it's more like the other way around. Because I have a leadership role with the user, I will do prototyping because I'm free to do so. The causal relationship is the other way around.

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

I think in most of my experience they [users] want me to take the lead [in presenting system options]. But, I also tend to work with relatively inexperienced users. I also think, especially in PC software, especially in the custom or small vertical market PC software, the state-of-the-art that's out there in terms of user interfaces is very poor. So that, it's very easy to suggest things that are better than the users would have thought of from looking at other things, and they're usually grateful for that, and they very quickly become spoiled by it.

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

...I don't know if the prototyping methodology has matured enough. Traditional methodologies came into being in order to try to make order out of the chaos that existed before, and I think it would be a big mistake to throw it all out and replace it a new chaos that has a fancy buzzword, prototyping. In a sense, some people would claim that that's what we used to do, we used to do rapid prototyping because we used to have no methodology and the programmer would just sit down and start coding, and look at where that led. I don't know if anyone has done any research to prove that what we have now is any more productive, because it seems to me that traditional methodology, although it has all of its proponents, seems to fail most of the time, more than it succeeds.... It seems to me that a carefully organized prototyping methodology, where you don't hold off on implementing. The idea that you have to hold off on implementing anything until you've designed everything, seems to me to be a major mistake. And I think I've seen evidence of it... I'm talking about rapid prototyping as part of the software development process, internally, as much as external design...

EXHIBIT X

Transcript of Responses to Selected Interview Questions

Respondent 22

Would you now share with me your own operating definition of prototyping?

I would describe prototyping as a solution to a problem that may not already exist exactly as the needs dictate. In a lot of cases, a canned, so to speak, package won't present everything that's needed. And in that same, you may turn to another device with which to accomplish your solution without modifying something that's already out there, a canned program. And, as you [to interviewer] describe it, it would be something that you that, before you were satisfied, it would be displayed [to user] and feedback received.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

Detail in the reporting provided... [Interviewer probes:] I would have to say that's the main goal.

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

The fact that they [unsuccessfully prototyped systems] were being integrated with something else, a more-or-less front-end product.

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

Again, you need the feedback. You put the example out there for them [users] to have.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

Depending on the application. [Interviewer probes:] An accounting system would be more costly, to set up a general ledger, accounts receivable. A custom invoicer and a specialized inventory management and tracking system would be less. [Interviewer probes. What makes accounting systems more costly when prototyped and other types of systems less

costly?:] Because of importance of the data... A great deal of care needs to be taken with something such as an accounting system. With the other, the numbers aren't as integral. It's just a small part in the accounting system, so to speak, when I mention something like an inventory [management system] with a specialized tracking system.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

[Not asked.]

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

Prototyping [is more effective at eliciting user needs.] [Interviewer probes:] When you put out a prototype... [it's] something for feedback. You're going to get feedback, you're going to get your ideas [evaluated]. Especially when you give them [users] time to work with it without your presence, they're going to tell you things that happen. In the long run, that kind of feedback is going to allow you to design a better product.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

I would say actual computations [should be included in the prototype]. [Interviewer probes:] If the real thing doesn't work, what good is it? It's the ultimate test.

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

I would disagree with that [recoding for executional efficiency] simply because I deal with it on a PC level, where it's more-or-less going in and working [on it] and reconfiguring and enhancing.

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

First, of course, service is important. And, you can just always take the annual report and see where the company stands there, the P and E [price-equity ratio], see what they're getting into... [especially concerned with research

and development expenditures]. [Interviewer probes. What is the impact of using R:base, with a small market share:] How do I feel about that? Again, go back to where the money [of the vendor] is going into. Is it going into research and development or is it going into marketing? In the micro-[computer] world I would have to say that marketing can strongly influence market share out there in the hand of the end users...

What would it take to cause you to leave your prototyping tool and move on to another?

I'd have to say some kind of revolutionary development in another program. Or possibly, difficulties that I've encountered in this current prototyping tool [R:base System V] which is facilitated...

What features would you desire in a prototyping tool that are not currently available?

Artificial intelligence.

What shortcomings do you see in the prototyping method?

I would have to say applications, prototyping allows you to become specific to the hardware I'm going to use it on. I'm only as fast as my slowest link... A payroll system done in R:base System V is going to be extremely slow on an [IBM PC] XT, whereas [on an IBM PC] AT... disk input/output is going to be important.

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

[Not asked.]

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

Yes. [Interviewer probes:] Substantial experience always... to me, experience speaks loud. You've crossed a lot of bridges already.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

The tool was here. The cost of my time to the company would have been about the same [unintelligible] and it would have been better for me to develop the prototype.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

In the case of Express technology [R:base System V's program generators] it's pretty easy to throw the menu together until you start working.

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

[Not transcribed.]

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

[Not transcribed.]

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

Some people call it vertical market, specialized applications it can provide the 'perfect' solution. Prototyping is going to come to a point where it's usable produce but if you leave your prototype or your project open for development later on, that's going to provide more of the benefit for the end user. [Interviewer probes. When is prototyping not an appropriate development method?:] Yes, I don't know if I can identify the markets, but off the top of my head I'd have to say there's some things I wouldn't get into [prototyping]. You're reinventing the wheel. No reason to prototype. There's a lot of room for prototyping but in a lot of cases, it can be reinventing the wheel.

EXHIBIT Y

Transcript of Responses to Selected Interview Questions

Respondent 23

Would you now share with me your own operating definition of prototyping?

I go through a process, whenever I develop a new system, which is... I call it experimentation, [which] probably is similar. What I will do is look at the requirements, put together a basic system that meets the minimum requirements, and then I will bring that to the customer, show them what it looks like. They [clients] will always have comments, things that are missing, things they don't understand, whatever. I'll go back [to the office] and come back [to the client] in another couple of weeks, three or four weeks with all the updates and show them the new changes. So I would call that prototyping. Prototyping is, I just basically analyze their input and output requirements, basically their output requirements and from that I look at the current manual system, look at the inputs, and everything is really geared toward getting the outputs [clients need]. Then I draw it up on a big sheet of paper, my own really messy drawing, to find the files, and then go off and do that basic working system. I would to like consider it prototyping because I don't have a real formal spec [specification] or spec review process.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

I would say user-interface a very key one [common characteristic of successfully prototyped systems]. Comfortable or efficient... a very good fit between interaction [and user]. [Interviewer probes:] The [common] characteristic is that those system do have a strong [user] interface, in other words, because of the prototyping the operators are more able to communicate more efficiently with the program.

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

There are situations where you do get into larger organizations where you need to communicate between different people working on the same project. And, you really are forced into a more formal approach, a less prototyping approach, and if you tried to prototype those systems exclusively, I think you would get to a failure situations, part A doesn't work with part B... And, not so much the size of the client organization but the size of the project itself...

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

The main reason is there's less overhead involved with prototyping, much, much less overhead. More time spent on actual development, I don't mean just programming, but more time spent on development, and less time is spent on the administration [of the project].

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

Again, it's all administrative... An awful lot comes down, I think, to if a project can be designed by a single person who has a very personal understanding of the application. I think that's where prototyping works extremely well. There's obviously projects that are just too big for one person, and I would think that's where prototyping would have some very major difficulties. Maybe you could have a combination obviously of some prototyping, some administration tied all together.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

[Not asked.]

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

Well, I think you should have some of both. You definitely need a very good data or information collection process in the front end of the project, whether it's formal or prototyping. But I think the user feedback on a shorter interval allows you basically to respond better to the feedback.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

Well I think the computations should be there, and the reason is that the users can see the results and give feedback on that.

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

dBase III [respondent's prototyping tool], the language itself, is an interpretive language, which is very good for prototyping. You sit down, you get an idea, you try it out. Once I get something working, and it's reasonably good, then I compile it [using Clipper, a compiler for dBase III code]. The dBase III itself under [PC DOS, the operating system] runs too slow to be viable product, but once it's working then I compile it.

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

What I care about is the standard, just like the IBM PC is a standard. If I buy a clone, I don't really care if they [clone manufacturer] go out of business, because they meet the standard. In the same [way], I'm really not too worried about dBase anymore, because that's a standard now. If Ashton-Tate goes out of business, it really wouldn't affect me greatly because someone else would be there with that standard. And I don't care if Clipper [a dBase III compiler] goes out of business, really, it would have no affect on me.

What would it take to cause you to leave your prototyping tool and move on to another?

[Not asked.]

What features would you desire in a prototyping tool that are not currently available?

[Previous discussions had revealed multi-user capabilities under the Unix/Zenix operating system.] ...Things like editors with syntax checkers. Multitasking kind of applications... where you can run your editor simultaneously with your debugging environment, and pop back and forth between the application and the code...

What shortcomings do you see in the prototyping method?

Assuming it's my method of programming, one area that's very serious is when you start to deliver a product and you're still prototyping things on it, because, what happens is you're getting ten different pieces of feedback from ten different users... so you plug in this and you plug in that and you end up sort of out of control because you are not really taking a lot of time to digest those requests and then line them all up into a whole formal update and release to all your customer base. I do maintain a revision con-

trol, so everybody eventually has the same revision, but it's a problem. Prototyping after the product is still in production... when the prototyping keeps working after the product is released, then you can get into some big problems... I can say that. You've got to keep the product within a very small sphere of maybe two or three users while you're doing the prototyping. What tends to happen is, you don't just sell one and prototype it forever. You sell one and they [users] work it for a while, and someone else hears about it and they buy it, and now you're prototyping this guy's [first client] and now you're prototyping that guy's [second client]...

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

[Not asked.]

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

Well, yes... a couple of things I do. When I come into a new client or a new application, I'll take stuff I've already written and show it to people, because it's similar. And, if they like it, I can prototype it some more. [Interviewer probes:] The more the developer knows about the application, the more he can use himself for feedback. You don't really need the... You know what's important so you can do it about 90% of the way and deliver the product and get some feedback on it. So, absolutely...

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

The client has a very cloudy or hazy... not really defined picture of exactly of what this thing [system] is going to do when he gets all done, or how he's going to use it. He knows, gut feel, he needs this, really bad, but as far as exactly what is it going to look like, is really up in the air right now... Since this project will require some groping to decide what data items I need, their [data item's] source, the trouble to get them and how to set up the reports, I recommended that we do this [prototyping]...

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

[Not transcribed.]

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

It strikes me that it would probably be less of a leadership [role], less often. Maybe where our premises are a little different, I'm not sure, I'm thinking I try to let the people [clients] who really know what they're doing take a leadership role and I try to follow what they're trying to do.

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

I think initially they [clients] want me to take the lead. They sit down and they're looking for all the answers. But, after... a while, they really get into it... and tell me what I need to do.

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

[Not transcribed.]

EXHIBIT Z

Transcript of Responses to Selected Interview Questions

Respondent 24

Would you now share with me your own operating definition of prototyping?

Well, it's [prototyping] an odd term. I think it would probably agree with the definition you've given [refers to interviewer's introductory definition], where it's a... instead of developing something and having it be an end package and upgrading it once a year, that you deliver something and based on their reactions you change your plan. You may be going from point A to point B, but halfway there the client says 'Well, that's not really going to work, now that we've seen what it looks like'. And, we just change the path.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

Most of them would be database applications, that I've worked on.

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

The only unsuccessful things I've seen in prototyping are accounting applications that had some sort of financial basis to them.

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

[A completed prototyped system, as opposed to initial delivery] is delivered faster. [Interviewer probes:] You have the interaction with the users that allows faster development, I think. Whereas if you're trying to satisfy everyone with a generic package that you're going to market then you have to try and satisfy everyone and you have to think of everything possible that they could want. Whereas if you're only satisfying one or two people...

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

I would tend to say yes [that prototyped systems are delivered at less cost than traditionally developed systems because] the development time is less.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

Probably equal quality. [Interviewer probes:] From what I've seen in the systems I've been involved in, people are willing to take sort-of off the shelf packages, that hasn't been developed in the prototyping manner, and they're willing to adapt it to their uses. Whereas the prototype system, it fits them exactly, it's not something that they say 'OK, well we'll change this so that it works on the computer' as opposed to saying 'The computer will change so it works with us'. And, I suppose depending on the application that could be a better thing but I think the quality is pretty much equal.

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

I think if I was going to be developing a package to be marketed to the masses, I would do a lot of prototyping beforehand. I think the prototyping does allow you get more in-depth into what the user really needs. Whereas if you're designing something to market to the masses, you're not really going to be that concerned with every... the smaller needs of the user. [Interviewer probes: does respondent mean that prototyping is more or less appropriate for a mass-market software package?:] If I were going to be developing what was going to be mass-marketed, I would do a lot of prototyping beforehand, so I would be more likely to prototype.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

I think actual data should always be only to test out what actually happens in the system, to make sure that everything is working correctly.

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

I think that [need to recode for executional efficiency] depends on how well you code your prototype, but yes, recoding is probably going to be necessary for the final product. But, I don't think.... I think it would be less work than coding from scratch to use the prototype as a structure.

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

Well, Ashton-Tate [developer of dBase III, respondent's prototyping tool], when I acquired it [refers to dBase II, predecessor product to current prototyping tool], it had a very good reputation. And the mere fact that they've gone from a [dBase] II to a [dBase] III to a [dBase] III Plus has shown me that they're a fairly stable company, and they've been acquiring other software companies as well. The stability of the company is very important to me, and I believe that this particular company [Ashton-Tate] has that.

What would it take to cause you to leave your prototyping tool and move on to another?

It would take, I believe that somebody would probably have to show me that another product is equal to what I already have and that it's a better or faster product. In fact, I've got some people trying to convert me over to another database management program, R:base, because they say it's faster and easier to work with, and I've been considering looking at it.

What features would you desire in a prototyping tool that are not currently available?

I don't think there really is anything that I've been wanting that I haven't got. There maybe things I don't know about, but...

What shortcomings do you see in the prototyping method?

Occasionally there can be user dissatisfaction. If the process isn't explained fully in the beginning then when something perhaps doesn't work, or doesn't work the way people think it's going to work, people become dissatisfied and it becomes a block, because once they are at the point [dissatisfaction] it's very hard to get them satisfied. I think that would be one of the major problems that I can see with it [prototyping]. It just takes a lot of explaining up front.

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

I think that the correlation is... Our company always charged by the hour for development. We charged for development time and in-house showing it and training it and all of that. And if we were going to be doing something where we just going to present and end product and drop it and do a little bit of training and that would be the end of it, I think the costs, the cost to the business to lay out cash for programmers, etc., would be very high and the end cost of the product would be high in the beginning until you had your costs paid for. I don't think because we prototyped that the cash flow or anything like that was good or bad. I don't know if there is a direct relationship there, but I know there is some [relationship].

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

I think that that could substitute a lot, because I could see what a prototyper could do is develop a prototype a much faster. Having been through the application before, they could probably second-guess what the user needs before the user thinks of it themselves. [Interviewer probes: In what way can't experience substitute for prototyping?:] Well, not every application is the same, when you're doing this sort of thing. At times, it is hard to guess what the user really needs, they didn't tell you something but when they see it, they actually say they wanted something else.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

It was something they needed immediately. They needed the bare-bones system immediately. It was basically to put out a fire that they had in the department, an extreme backlog. So, we put the bare-bones system in, and now what we're doing, is we're constantly refining based on what we find from the bare-bones [system], and the bare-bones only took six weeks to develop.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

Probably more anxious. I'm always nervous dealing with a new project because you are dealing with unknowns, as far as how the people [clients] will react to this prototyping process and things like that. And, it's kind of difficult to gauge until you're actually in there how they will react to the prototyping as opposed to you just dropped a package in.

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

Probably more often. [Interviewer probes:] Well, in the prototyping, when you go back and say 'OK, what didn't work this week or what problems, or what enhancements do you want?' you sort-of have to draw it out of them. Sometimes people [clients] are really reluctant to bring up things because they think they're too minor a thing to discuss. So, you sort-of have to draw things out of people as to what they would like and so you have to sort-of prod them and ease them and things like that.

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

I would say it's about 50-50. Some people [clients], you suggest things and they say 'Gee, I never thought of that' and some people are just filled with ideas themselves. It really just depends on the people's personal backgrounds, what their exposure to computers is and what their exposure is to the field that is being worked on.

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

This whole process is something that I haven't really thought about much, but I've always have been doing. It's just something that I find very interesting, that you'd [to interviewer] be studying it... I think I'll probably most interested in finding out what people's [other consultants] reactions are, what people are concerned about in the [prototyping] process, because I know that I've had some resistance to it, where people have said 'Look, this isn't working, we can't continue like this, it's just too much of an interruption to the business' and things like that. And, I'd be interested to see what the general reaction is... It's something [client resistance] that I've had, but not a lot. It just depends on the people, how much they know about computers, and about the business... [Interviewer probes: Does prototyping affect cash flow and other finan-

cial considerations?:] I think that most of my clients called me because I was notorious for getting systems in fast, to solve immediate problems. Most of my clients had immediate problems that needed to be solved... And, that something that lends itself very well to the prototyping process, and it is true that if you went in and analyzed a problem and developed something and then dropped it in a year later, people would just say 'What's this? We've forgotten all about it. Our problem is so much worse now that we need something completely different'. Things like that. And I think it probably is a process [prototyping] that does help some businesses that do both. [Interviewer probes: what differences does respondent see between being an independent entrepreneur and a corporate staff person?:] The only difference is in my state of mind. Working for myself, I worked about sixty hours a week. As I said, 80% development, 20% administrative. And the extra ten or fifteen hours a week that I had to work was mostly just to do the administrative work. And, in the corporate environment, you are less pressured, I think, to get things done. It's a much slower pace...

EXHIBIT AA

Transcript of Responses to Selected Interview Questions

Respondent 25

Would you now share with me your own operating definition of prototyping?

[Following presentation of interviewer's introductory definition:] Well, the only thing I'd take out is our operating definition [of prototyping] is that I don't think we would intentionally give something to the client that we knew the client wasn't really looking for or had not already been discussed, in that we start the design process with the analysis phase, where we try to get down on paper, a paper prototype of what the system will look like. And, on paper, take the user through various screens and walk the user through how the system is going to behave, how it's going to perform. At that point, they have a chance to interact with that, make changes, etc. And at that point, once those changes, back and forth, takes place, then what we do is we finalize at least the paper design of the product. And then move from there to a computer design, a prototype, if you will, where we make our best effort to match what was on paper on the computer. And, that's what we show to the client...

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

Typically they're [successfully prototyped systems] are database management systems. More than one user, not necessarily a multi-user system, but more than one person responsible for data entry or reporting, etc. And a function that was relatively high on the strategic importance to that area, to that department or to the company or whatever you define as the area that the application will be used in.

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

They [unsuccessfully prototyped systems] all had to be done yesterday, when in reality they really didn't have to be done yesterday. So that the impression... They also did not have the proper resources allocated to the system, in other words, there wasn't a direction from the top, that this was an important system, there wasn't the proper funding, etc. And three, we never really pressed our customers to do it. And we learned from that.

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

It's just that you never have to... Well, the amount of time you spend backtracking, going back into the system to add the features that you forgot, that were not included, or that the customer said 'Gee, wouldn't it be nice, now that I can see what it does, wouldn't it be nice if we could do this'. So, you're always going back, rather than taking care of it at the front end [i.e., prototyping], you would have saved yourself a lot of time and a lot of energy.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

I would think it's the same cost, because the amount of time you're spending at the front end [prototyping] is about equal to the amount of time you spend at the back end, backtracking and adding [i.e, perfective maintenance]. So it would be equal with us, in the long run.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

Well, they're [prototyping systems] are well thought out. There's a definite structure to them, there is a definite road map, the road map is a logical road map, and the pieces fit together quite nicely. And, the various hooks that are left that you need for future expansion are there, all in the proper places.

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

I think the traditional method of writing an application is fine, as well as I think prototyping is fine. I think you've got to combine the two. I mean, there's a place for prototyping in the traditional method. One is not done at the exclusion of the other. So, you can have your flowcharts, you can have your analysis, you can have your requirements document, and you can have your prototype at the same time. Because you've got that [the prototype] as an interim step... I think that [the prototype] is an integral part of it [systems development process]. There's room for both.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

If you had that spec [specification] document, that document would say 'This calculation is as follows'. And, while the prototype wouldn't have to go into performing that calculation, you would at least have a document that would explain how that calculation takes place. I don't think you want to run off and do prototyping at the exclusion of doing the analysis, because you're going to wind up spending, wasting a lot of time.

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

It [recoding for executional efficiency] hasn't been our experience, it really hasn't been our experience...

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

It's really market driven. We're a market driven company, so we're not at the vanguard of...

What would it take to cause you to leave your prototyping tool and move on to another?

Massive market demand and support, or even just a showing of market demand...

What features would you desire in a prototyping tool that are not currently available?

Better code generating capabilities... I think we're seeing more of that, I can't say I'd like to better [sic], I can say I'd like to see the path [of development of code generators] continue...

What shortcomings do you see in the prototyping method?

Perhaps time might be a shortcoming, in that it does take longer, at the front end, it takes longer to get something into the client's hand in the front end that's working. Two, perhaps an inordinate amount of time spent in design, the feel of the program and the features of the program. You end up working on an unfinished painting. There's always something else to be done. Our primary approach is that there is an opportunity to be taken advantage of, where there's a problem to solve, so let's solve it and now let's get on to the next one.

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

I would say it's a very strong relationship [between prototyping and financial operations of the business]. It's not a direct connection. [Interviewer probes:] Well, if we pick the wrong tool, we end up wasting a lot of money and have to eat that time because the client's not going to pay for it. So I would say it's a medium-strong relationship.

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

I think it helps out on the other end, where you're performing the applications, where you're writing the specs [specifications] and writing the applications document. I think the system still must be prototyped, but I think the amount of time you spend in understanding what the application is going to be doing, what is should be doing, what are the calculations that are going to go on, I think it's [previous experience in developing a similar application system] cutting down on that time. But I think you're spending the same amount of prototyping time. You must have a level of understanding of what the application should do, then you can go off and prototype it.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

Because of the size of the project and because of the complexity of the calculations that were going on and the various things that were happening inside of the application.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

Because you can see it [the system] taking shape. Your understanding of the application changes. It's better. You can put your hands on it, you can actually... All those discussions, all those interviews are boiling down into something that looks relatively simple. I think it tends to lessen the anxiety on your side and lessen the anxiety on the client's side. The client's paying for something and they can actually start to see it.

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

It's the same. As long as you are up front to the client, what are the steps going to be, and the client knows that this first stage is a prototype, it's not the working application, and we're going to be going back and forth for a while, that's fine.

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

They [clients] do [bring up system options], often times they do, but they often looking for you [the developer] to take up the lead, so to say.

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

It's just that we don't view it [prototyping]... We're not an MIS Department so that our actions, our interactions with the user are really as an independent third party. And number two, we don't really view the prototyping as a tool. It's a very integral step in how we solve a particular problem or take advantage of an opportunity. It's like saying, you've got the requirements document and then you begin working on the prototype. You don't leave out that step, and that step more and more is becoming less and less of an optional step... It's becoming more and more a required step. That's our approach. [Interviewer probes: returning to relationship of business operations and prototyping:] I did mention that the client feels happy, but I didn't tie that into the financial [aspects of the business.] Our clients are billed regardless. You like to hit a client with a bill when they feel happy about it, and prototyping is one way of making sure that the client knows you are working on the assignment. There are other ways of doing that: we have regularly scheduled meetings, we have progress reports, we have all those kinds of things... I can see where people [developers] would say that [there is a relationship between financial operations and prototyping]... There are other means of communicating the point that progress is taking place, to make the client feel good about spending all this money.

EXHIBIT AB

Transcript of Responses to Selected Interview Questions

Respondent 26

Would you now share with me your own operating definition of prototyping?

My definition of prototyping is very much what yours is [refers to interviewer's introductory definition], which is a pleasant surprise since I've come across so many [definitions of prototyping]. Essentially, for me, prototyping, we tend to use the word iterative development instead of prototyping just because there is so much confusion about the word prototyping, but when we're doing what we consider to be the most valuable method of prototyping, iterative development, it's a matter of very quickly fabricating essentially a shell, the visual appearance of the system, which is able to demonstrate what the functionality of the system will be ultimately, without necessarily containing the code to actually deliver that functionality. The primary motivation for that is, as in your definition, to get a reaction from the end user of the system, to find out if this is what they had in mind. Typically, I'm sure everybody in the systems development world knows, what people [clients] ask for and what they want aren't always the same thing, and this [prototyping] is an effort to 'guide' the systems development effort towards what they want, as opposed to what they say they want, as well as to manage their expectations and their understanding of the process.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

I think, first of all, they are highly interactive systems. Nobody really wants to watch a batch job [be] prototyped. I think, generally speaking, there has to be a good political relationship between the developers and the end users. And, there has to be a willingness for the end user to get involved in the problem [solving] process. And if all that is true, then I think it [prototyping] can be successful.

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

Probably mostly of the opposite of what I just said [refer to response to success question]. Namely, that there was perhaps an adversarial relationship between the MIS area and the end user. Or, the user was not able or willing to extend the effort required to be involved in the [prototyping] process.

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

I think that, in my perception, what you're getting from prototyping is largely quality and accuracy and satisfaction with the ultimate product. I, generally speaking, don't think it's [prototyping] any more efficient than any other [development] method, from the point of view of calendar time or from the point of view of effort expended.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

It may be more expensive to develop a prototype system than a traditional system. And some of the reasons... One of the biggest reasons is that the users tend to get more of the bells and whistles on the system because they have a lot more time to lobby for them, and they become a lot more aware of the value of an enhanced user interface or what have you. And those things take a lot of time and cost a lot of money.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

I think largely that the value of prototyping is that one allows the user to end up with a system that meets their needs better than a system developed using traditional methods. They [users] have a chance to control the direction of the development effort.

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

I that prototyping is probably better at getting at their [users'] needs. I feel that most users, especially naive users who have not worked in [systems] development before have a great deal of difficulty articulating even when being interrogated by an expert systems analyst, have a difficult time articulating their [unintelligible].

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

I think that people who get involved in that sort of controversy or exercise are thinking of the prototype in the same way that people look at the ultimate system in the traditional development method, that is, that there's one prototype, that you undergo development effort to deliver a prototype. You ask the users how they like it, and then you go off and build your system and I don't see that method as being all that much different from the traditional method, it's just one extra step thrown in there. I think that first version that you show the users probably won't have any underlying computations because you want them to see something [a prototype] as soon as possible. On the other hand, that doesn't mean you don't show them anything again until the end. In this case, I'm talking about what we would call a constructive prototype and not a destructive prototype, a prototype that is going to evolve into the system itself.

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

I think the people who are throwing prototypes away are not employing rigorous development methods on the prototype system, and therefore feel that when they... I think that if you want your prototype to turn into your system, and I feel that, generally speaking, that's the most effective way to prototype, you have to use very formal and consistent development methods, the same methods you would use when you are developing a traditional product, so that in fact a prototype doesn't mean a poorly developed version of the system, it means a well developed version of the most visible parts of the system, those that would be most effective at eliciting a response from the user. If one is just slopping something together as a so-called prototype, one certainly doesn't then want to sort of turn that into the system because if you start off badly you're going to end up badly. But, if you're very rigorous from the beginning, I think it's possible [to evolve a prototype into a production system.] [Interviewer probes: what about executional efficiency concerns?:] I would say that it's possible that some recoding may be necessary for performance reasons, but I also think a lot of people are too hung-up about performance. Depending on the type of system you are developing, performance may not be an issue at all. Again, if you're rigorous in the use of tools from the beginning, that [recoding for executional efficiency] shouldn't be necessary... [Interviewer continues probing:] I also think a

sound design, which should be an element of the prototype, that is, I don't think prototyping replaces the process of the design of the system, and that a sound design will lead ultimately to a sound system, and if there is performance tuning needed at the end, then you do it at the end if you need to, and if you don't [need to tune for performance], then you don't worry about it.

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

You try to find the most stable and the most promising firms. That is, the stability of the firms is one of the principle factors in our choosing any software that we're going to use in the development effort. [Interviewer probes: how is stability judged?:] Word of mouth, reputation, length of time in service, past successes, frequently we'll get references. If we're involved in choosing a very critical piece of software, then we'll do reference checks. We'll get reference from [consulting business] competitors, we'll talk with as many people as we can, ask as many difficult questions as we [can] about it.

What would it take to cause you to leave your prototyping tool and move on to another?

Generally speaking, if I'm in an environment where I have a tool that is adequate for the job, I'll use it. That is, I won't incur a learning curve unless I have to because our client's paying a lot of money to have me learn tools, or else we're paying a lot of money, one way or the other. So, if I don't have to switch tools, I tend not to, but if I move to a new environment, where the tools I had been using are a different environment, and when I say 'environment' I mean hardware and/or software, operating system architecture, then there will have to be an effort expended to select and learn new tools, and generally that goes into the estimating process for the project.

What features would you desire in a prototyping tool that are not currently available?

That's hard to say. In any given environment, there are inadequacies with any set of tools. I'd say in the micro-[computer] environment, there's a lack of cohesion between some of the screen handling and file handling capabilities, unless you get into fourth generation languages, which on a PC tend to be quite slow and quite limited in their quality as development tools.

What shortcomings do you see in the prototyping method?

I think it [prototyping] can be expensive and I think that it needs to be done with the right user at the right time. That is, I don't think it's a panacea, I don't think that

situations dictate prototyping to be used, although I'd say, in a sense, it would be my default value, that is, I would choose prototyping unless I felt circumstances dictated against it.

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

I think by being very good at prototyping, we develop better systems for our clients, and the better job you do the more money you make. As I said, it's not a one step relationship, it's a two step relationship, that prototyping improves the quality of the work and the quality of the work produces the financial condition of the company. [Interviewer probes: what affect does prototyping have on day-to-day financial operations:?] I'm not sure there is an effect.

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

I don't think that is [substantial previous experience] is a substitute, in fact I'd say it's almost a good argument for producing a prototype because you can probably produce a higher quality, a better prototype more quickly and demonstrate your knowledge to the client. And make sure that this is really what they want. That is, I don't think arrogance or presumption on the part of the developer is a substitute for prototyping.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

We chose to prototype the project because we felt that the way we were planning to solve this particular user's problem was substantially different from previous attempts to solve their problems. And, we wanted to make sure they have grown to be comfortable with that solution before we finished it. They have several abortive attempts... several unsuccessful attempts to develop a system that would meet those same needs, none of which adequately performed.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

There's a lot less fear of being wrong in a prototyping environment. If I go back to a client three weeks after we set up development and showed him something that he doesn't like, that's going to be a lot better than showing it to him three months after the project started. He's going to forgive me for not knowing what he wanted so early in the project and say 'I didn't express it right; here's what I wanted'. But three months later, you don't have the same excuse. At the point, you can talk to the person [client] a great deal, a lot of time has elapsed, anyhow by then they expect you to do the right thing. So, if you ever did the wrong thing, and it showed up very late in the game, I think there's a good deal of risk there.

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

I'd say it's the same. Generally speaking, we tend to take a leadership role with our clients, and guide them through the development process. The user's more involved but in either case I think we're trying to [unintelligible]...

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

Generally speaking, they expect us to take the lead, although there are people [clients] with a great many opinions, and we're always trying to combine their needs with our expertise.

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

[Interview requests expansion on "right user at the right time" point:] I think the right user is a user who wants the system as opposed to the user who has been told they need to have the system, and I think what you get from that is a cooperative user. You have a user who has a great deal of stake in the quality of the system. And, a user who's willing to think hard about what it is he wants. Someone who's not so busy at trying to keep his head above water that he can't think about what he really [wants] needs longer time; users who show up at meeting and play with the prototype and [are] sincerely interested in the ultimate quality of the system... They have the luxury of spending the time necessary to get the system that they want, as opposed to essentially delivering them one very quick version of what's supposed to be the final system, that is, they're not trading off time at the expense of quality. I think there can

be a trade off there: 'Give me 50% of what I need quickly, and go away and don't talk to you any more'. There are people who build systems that way and they end up with 50% of what they want... The right time is when the user has decided he has the time and the commitment and the support to get involved in the project. [Interviewer asks for further comment on the evolutionary versus modeling question:] We tend to focus on evolutionary development. Modeling, it's hard to justify spending a lot of the money and time on thoroughly modeling the system, when for a comparable amount of money and time, one can develop significant portions of the ultimate system. But in order to do that last part, there has to be... you have to be developing in a highly productive environment, and that productivity comes from a knowledge base and experience, having very good, strong people which our particular company does, so we're able to... we have formal methodologies for the way [unintelligible] and within the scope of these methodologies we're able to develop effective evolutionary prototypes very quickly. If that were not the case, then we'd have no alternative but to use modeling prototypes. If it took two months to deliver the evolution of the system, then I guess we wouldn't really have a choice, but considering that we can probably develop an evolutionary beginning of a system in several weeks, its worth having the client wait another week or two before they seek to see something that at that point is fairly realistic. [Interviewer probes:] And it does require high end resources... mostly high end people and methodologies... you have to have the right people doing the right things and knowing what they're doing. I don't think it [prototyping] would necessarily work in the large MIS shop that had essentially average level MIS people in it. I think a small company like us is very effective at it because we can invest people, we are very focused at what we do. It's a very high-energy, highly motivated environment, and that's why we're able to turn things around very quickly. [Interviewer probes the relationship between financial operations and prototyping:] I think that the quality of the work is very important, as I said before, we're managing the risk of having the user involved in the process and know what they going to get what they want, is important for client satisfaction. But we haven't had a problem with clients withholding payment to solve something. We tend to conduct business at a very professional level, and that just tends not to be a factor. If it does, it's probably not the client we really want to do business with anyway. So, it [prototyping] keeps us in business in the sense of it allows us to fulfill the needs of the client. As far as the month-to-month cash flow, and getting them to sign the P.O. [purchase order] or sign the check, if that's what it takes to get them to sign the check, then we're probably doing business with the wrong people. [Interviewer probes:] Cash flow is crucial, but cash flow doesn't necessarily hinge on prototyping. In the sense of, if I were one-shop operation [a single entrepreneur], or rather if I were hiring a one-

shop operation, I'd be much more reluctant to let go of any money until I saw some results. When hiring an organization such as ourselves, where we have an industry reputation and we have a certain level of integrity, strong reference, companies tend to be less ill at ease with the process up front. I think we also have a very skillful management staff and there's a great deal of focus at our end both on managing the client and the client's expectations as well as building systems. That is, we're not just a bunch of programmers or hackers, we're really focused on being a full-service organization and, in that sense, we provide a lot more feedback than just the programming side.

EXHIBIT AC

Transcript of Responses to Selected Interview Questions

Respondent 27

Would you now share with me your own operating definition of prototyping?

First of all, I'll address prototyping as you have defined it [refers to interviewer's introductory definition], which I think is very traditional. I think that the traditional approach to prototyping is going down the tubes, except in the financial area, and even in that area. Prototyping as you have defined it was a requirement that we had to have when we had programming languages like COBOL and BASIC and even before those... We had to work with the users and try to define what we had to get a specification, try to freeze the specification and program it. And, then go back to the users and say 'Aha, there's your beautiful system' and the users would look at it and say 'This isn't what I intended to do'. The problems were fraught with a lot of things, a lot of perils in that period of time. One of them was the inability of getting the users to define what they wanted when so many of them were in their first management position, who didn't know a heck of a lot about management, and they're telling you how to do something to develop a management system. So, as a result, there was a hit-and-miss type proposition, and the problems were the languages didn't lend themselves to change. You froze a file, you build a big flat file, you tried to dump data into it, and all of a sudden this guy would [want modifications]... And you had to go and undo those programs, and you were fighting the problems of opening files, and closing files, and how many files could you bring into the computer?... Now, when we get into the fourth generation world, we started operating more like a company operates... All I've got to do is write a program that maps information and [works at a high level]... When that type of a capability came along, we could go to users [and easily modify data structures, especially files]... So, we could work earlier in the cycle in terms of writing programs. We didn't have all that garbage we had to write that is attendant with COBOL. Therefore you can go to a user earlier and say 'Let's make our mistakes together. Let's grow the system'. And that way, maybe nine months down the road, we can say 'We've had a false start...'. We've only wasted nine months as opposed to three years... So, I don't know if I've really answered your question. To me, the old approach to prototyping [respondent uses this term as equivalent to system construction], I'm brain washed, I'm a missionary for fourth generation, DBMS [database management system] approaches. I think it's the only solution. The [artificial] intelligence languages are going to take us further step forward, hopefully yes, but right now I think we've got all we can do. Finally, we're working with users

and we're putting something in place [that is] much more realistic.

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

I would say very strong project management. Knowledgeable, able to work with people, and top management backing. Those are the two big things.

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

I would say two things. Design from the bottom, and support from the bottom [were common flaws].

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

Well first of all, I have to answer that by saying [that] I think prototyping... We have to draw a line: today's prototyping, yesterday's prototyping, so I'm biasing the answer somehow, probably from your viewpoint. But, yesterday's prototyping [i.e., conventional development methods], under the right circumstances probably was halfway decent. Does prototyping reach the goal of successful installation sooner than conventional [development]? I'm going to make one assumption in my answer, and that assumption is that it's a well-managed project. I'll say yes [prototyped systems are delivered in less calendar time]. [Interviewer probes:] Because if it's properly organized, you get the user involved, and he understands the pitfalls of his contribution, and he works a little harder, everybody works harder...

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

It's the way you measure costs that counts. If you put a bad system in... First of all, assuming your prototyping does a better design job and the ultimate outcome is a better system, I would you put the investment at the front end and when you put it after the fact, to correct the system, you're pouring good money after bad, and I just don't think that makes sense. You always do some of it [corrective maintenance], but it minimizes it...

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

I would say greater quality. [Interviewer probes:] Well, any time you get the user involved in design, assuming knowledgeable users, you're going to be better off in terms of what your product is. Data Processing people are not normally managers. Systems analysts are few and hard to come by that are really good systems analysts. Programmers who knows what a manager needs are few and far between, they're mathematicians or very numbers-oriented people. So, if you get the user involved with the proper amount of guidance, I think you'll increase proper balance [of expertise].

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

It depends on who's doing the project... In consulting, I found that with consulting people, you could sit on the top, and look at the way something should be done. Prototyping, at times, will cause you to automate what is being done. Traditional development isn't going to solve the problem either. It's a question of the top level design down.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

I would prefer to have as much of the computations done as possible. I don't think that was practical in the old simulation programs. I think the new DBMS [database management system] approaches allow us to put databases together and make it real life, very fast.

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

I don't think in today's languages that's [recoding for executional efficiency] true. Possibly in COBOL days or something, that might be [true]... [Interviewer probes:] We've run Progress [respondent's prototyping tool] against COBOL programs on benchmarks, and we've run just as fast. So, the technology today is there.

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

I think they're [Progress Software, vendors of respondent's prototyping tool] in pretty good shape. [Interviewer probes:] First of all, I knew the four guys that started it, we worked for the same company. And when they spun off from this mainframe environment, they had already had seven years of writing DBMS [database management systems] languages on the mainframe. So, they turned around and decided to write it in the micro[computer] world. The fact that they have been able to port onto [a large number of computer systems], I think speaks a lot for it.

What would it take to cause you to leave your prototyping tool and move on to another?

The opportunity to sell my product differently than we're being able to sell it today. It would have to be a financial enhancement. I love the language.

What features would you desire in a prototyping tool that are not currently available?

Graphics... We feel that graphics are a major requirement... We feel that interactively having conversations with other database management systems is a requirement... They're, I think, the two major ones [desired features] that I run into. They're are others... [an applications generator, pending from Progress, respondent's prototyping tool]... The other thing... that I would like to see more off, is better remote training.

What shortcomings do you see in the prototyping method?

I would say right now is one of the biggest is lack of available Progress [respondent's prototyping tool]-trained [personnel] resources. In other words, we took it upon ourselves to jump into this fourth generation approach, none of us want to train people because of the risk, and I would like to be able to go down the street [and hire Progress programmers]...

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

I think the minute that you get into prototyping you're admitting to the world, you're admitting to the customers that you don't have a product yet, and so they beat you down in terms of the price, and you have to be prepared to make an investment in support. You're finding bugs in design, you're finding bugs in programs, you're trying to train on

the fly. Along with this, you've got programming language changes... So, I think it's a tougher road to go, I think it's tougher management-wise. I think it requires an awful lot of selling, internally, for that customer, and I think it takes some hard-nosed business relationships because what you do is, you've got people trying to change things. And your goal is to finalize a product and get it into the marketplace and accept it. And, when they [clients] continue to keep changing things, you're delaying your objective [introduction to the market]. What we say is 'Look, we've got the design, we're going to have an enhancement program. Let's freeze, and take these enhancements put them... aside as a requirement with a commitment [to upgrade the system] and let's go'. And, that has worked fairly well. [Interviewer probes: what about relationship of prototyping and cash flow?:] I think when you've finished prototyping or start getting to those stages where you have a confidence level that it's a viable product for the marketplace, all sorts of things happen, all sorts of things happen. You start worrying about the design of your training courses, how you're going to market, what your sales literature is going to be, and everything. Which, again, is another commitment that you have to make to it [the product]. Depending on your cash flow, in a small company like ours [one may not be in a competitive situation]... [Interviewer probes: questions specific cash flow problems and relationship to prototyping:] I'm saying you live with it, it's a way of life, we have to live with it, but what I'm really saying is when you do prototyping you're going to make a commitment to some money [to be expended in the future]. As I said before, you put your investment up at the front end. A lot of people design a system, plunk it out in the world, sell it, and then run into trouble and they spend their time bailing it out but they've got the cash flow coming in. We've been more organized and structured in the way we've developed [systems], now, all of a sudden, we're starting to sell but we don't have a lot of capital to throw into marketing tools, so we're going to initially grow slowly. But I think there's a cycle and there's a curve, a break-even curve type of thing where we'll regenerate flow cash back into these tools and then all of a sudden we're going to get the investment. That's been our approach.

<u>Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?</u>

Well, I think a lot, very heavily. I think the reason why our systems are as good as they are, and been developed in the time they have is because of my experience. And I guarantee you that where I've had experienced systems analysts working for me, my client situation has almost always been better.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

Because I think it's one of the hottest markets in the future, [field] repair maintenance.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

Well, I think to manage is to worry, but on the other hand, I feel comfortable because if I have powerfully [sic] picked the client, I'm getting a value in terms of design that, once I had the most experienced organization in the world, I probably couldn't have otherwise. So, it's a love-hate relationship.

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

Well, you're talking of the use of your [developer's] resources plus the use of theirs [client's] and you're performing a consulting role because they give you the user design, but you're [developer] the person that knows the language, you're the person who's trying to engineer their product towards a final end. And, to us that final end is to generate revenue across the board, and I do not want any of these people to back into a corner where I end up spending resources for a special requirement on their part unless they're paying me extra money to do this. So, you do have to watch it very closely.

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

I think that depends on who you are working with. If you're working with a company where they've got strong people, and I define strong in two ways, one is knowledgeable-strong and the other is unknowledgeable-strong, you're going to have a problem. If it's knowledgeable strong, you may [unintelligible] if you properly engineer it and manage it. It's a lot like consulting, from that viewpoint. If they're [clients] going to have strong people who are unknowledgeable...

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

[Interviewer probes: requests respondent to continue to address relationship of prototyping and financial operations:] Well, what we do, we enter into a contract whether it's prototyping or otherwise, we normally have a payment schedule of at least three parts... more of a progress payment type of thing. We essentially get money on the front end, and then we deliver something and we get money at that time, we give them fifteen days to accept it, or whatever is negotiated. And then, if it's going to extend on, we will get and have other project points going. It really depends. You see, none of our prototype projects have been over a year, and both of them that have taken a year, the last two anyway, shouldn't have taken that long. The thing that most people don't realize in a project [is] you don't control people on project as if they're your own people... I bring that up because we haven't really pursued other than the three- or four-payment type approach... [Interviewer probes:] Well, I think it simply boils down to is the objectives of the person who is customizing or prototyping for a job in-house... We're looking constantly in terms of what we're going to develop, where it's going to generate future cash flow. I think there are two things. One, we soak up everything we can from the users. We're like a sponge, as much as we can. We have to because, being a small organization, we don't have the resources all the resources to discuss things with. But, the other part of it is: once we do that, we then have to be more the managers of ourselves than when we're developing things in-house, and there is a difference. [Interviewer probes: queries respondent's use of "old prototyping" to refer to conventional development:] Oh yes, I've always been very structured. I worked for [a manager who saw the need for prototyping] but we couldn't get the opportunity, we couldn't sell management that they should do it... I think running development as a manager, one of the things that every manager wants to do, and it's part of information theory [decision theory?] and that is, how do you minimize risk? How do you get as much information as you can, as early in the cycle to make that information probability of decision-making higher? What we're really talking about is probabilities. The more you can do prototyping, maybe it's the application of 80-20 [rule]... you don't have to do the whole thing, you look for where the problems [are]. And, if you can identify that in the right environmental systems design, configuration type setup, you're minimizing the risk, and maybe that's all the prototyping you have to do... And whenever we do the prototype I always looking, every week, and saying 'Where's my most potential problem, and let's get that under control'.

EXHIBIT AD

Transcript of Responses to Selected Interview Questions

Respondent 28

Would you now share with me your own operating definition of prototyping?

I guess the strict definition pretty much agrees with yours [refers to interviewer's introductory definition]. It is not... really what we do. My definition is perhaps easier to look at from a hardware standpoint or some device, some widget, that you develop a prototype to try to sell it: try to sell the prototype, not necessarily what it looks like, in its final form, but here's the prototype of this particular gadget, and it doesn't work and now we need money to develop it. From the software side of things, it pretty much agrees with what you said. The only time that we would follow that strict definition would be if we were developing a product without a known customer, in other words, we had no one to sell this to. No one has come to us and said we need this product. But, we perceive a need in the marketplace and have the time and the money to develop some kind of a product. In this case, it would be a strict prototype via your definition. We would put together the way the system would look overall, with perhaps most of the screens, for example, developed. A few of the output reports in place. One of the major functions of the system probably fairly close to being at least our version of being complete, with the rest sort of hanging out there, tempting on the menu. And, at this point, we go into an initial sales, marketing environment. We would try to capture as many commitments as possible. Basically, if we were able to complete this to your specifications, Mr. Customer, will you commit to buying it if we do that? At the same time, arriving at some kind of a market price for it. Now, as I said, we do that very, very seldom. I don't think we've ever carried that to its complete fruition. We are involved in one now, in the very preliminary stages of putting together a prototype, but that is not our normal way of doing business. However, the so-called traditional approach is probably not our normal way of doing business either. We probably have a... the majority of the cases involve a somewhere-halfway-in-between kind of approach. The typical example would be the customer comes to us with a particular problem which is not solved by an off-the-shelf piece of software, and he would come to us and we would negotiate a contract with him to do this particular piece of software. We would get from him as much of the specification for that system as we could, from a general standpoint. It almost never gets down to the number the customer thinks... we have eighteen modules and fourteen screens and eighty-six reports, but this is the information he has, this is the information he needs, and how do we get there? And that's

his problem, and that's the way he comes to us. At that point, we go through a rather sophisticated home-grown formula to determine how many modules, of what complexity, how many screens, etc., etc.. And from that, develop a time frame and therefore a price. A time frame from the traditional manmonths standpoint, or mandays, and then perhaps also a calendar day from that based on our current workload, etc., etc.. And we go back to the customer with that and say, this is what it's going to cost you. From that point on, we would start developing the system, but we would typically interface at some fairly regular interval with the customer, at various points in the development of the project, so now we are in fact prototyping to a smaller degree, or to a little bit different degree. We would put together what might be the main entry screen, for example, two or three of the main input or record types that are involved in this particular project. And then we would sit down with the customer and show him that, and say 'Is this what you had in mind? Is it going do the job for you?'. We have certain set ways in which we design screens, that normally everyone likes. We would do that same kind of thing for the customer, basically because it saves us time to do that, because we have known code that produces this type of screen, we can to a certain extent fill in the blanks on some big parts of the software. So at that point, we get a reaction from the customer. Now this is not really a marketing reaction, the job is sold, the money is coming in, but it is a satisfaction kind of thing. Are we doing what he expects? Is he going to be very unhappy at the end? And, because of the way most of these contracts are written, he must be happy at the end or he's not going to pay the remainder of the contract price. He's going to find something wrong, that he doesn't like, and he's going to say 'It's not complete', and everybody's unhappy, so it does pay to do that kind of prototyping... Now that [prototyping] can be dangerous at times from the standpoint of, you can get involved with a customer who does not think the way you do... and wants a lot of changes. One of the things we attempt to do with any piece of software that evolves, no matter how customized it is, is we try to approach the writing of that software with the idea of selling it to someone else besides customer number one. We do not sell software, we license it like most other people do. So we license it to that customer for his use, but it's always non-exclusive. So, if you get involved with this customer and he is changing things such that it is making the product good for him but bad for everybody else, you can wind up up with a problem, as what do you do with this guy?

<u>Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?</u>

I guess my instant reaction is no... The user will typically get a better feeling about a piece of software that has been

prototyped via my definition. He also gets an appreciation for what you're doing for him...

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

No. There have been very few, fortunately, that have not been successful. So, I don't have a good base to judge from.

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

When you're looking at total customer satisfaction, to reach a particular level of satisfaction, I think the prototyping will wind up taking less time, although as you're going through it, it appears to be taking you longer, you the developer, longer, because you can always do it faster than having this meeting with the customer. But in the long run, you're much better off.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

I would tend to think that the cost is probably the same, on an overall basis, talking about strict monies passing hands. You're probably going to expend about the same effort in the long run, because when you don't prototype, you're almost guaranteed... At the point in time when you feel you're delivering this product, you will not be, you will be delivering something which has to be modified to a much greater degree than if you developed something with a prototypical kind of method. And now if you deliver it and if you still have to make changes, they are very few, comparatively speaking. So that the cost to the customer in terms of hours spent, which is typically what he's paying for it, is probably about the same. The gain for the [consulting] business, however, is much greater when you prototype because you have a happier customer in the long run and that's what pays off for the future.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

[In response to question regarding successfully prototyped systems:] The user will typically get a better feeling about a piece of software that has been prototyped via my definition. He also gets an appreciation for what you're

doing for him. An example of that would be that if a person [developer] who takes a bare specification and does a coding job, as opposed to an analysis job, is not going to be very successful. Whereas if the customer say 'I want X' and you don't question whether or not they need it or, what about the effect of Y and Z on X, Mr. Customer? Usually, because of this interaction with the customer back and forth, what you wind up is with a better system to sell to other people. It's also a better system for that customer because although sometimes you can look at this from a strict marketing standpoint and you just do what the customer says, and it's wrong, you know that he's going to come back to you later, more than likely, and say, 'We need this modification'. And if you think in terms of strict dollar signs, that way you're better off not opening your mouth. But, I have found that really has a negative impact, and you're better off talking with the customer trying to find out what he really needs, and that you find out during a prototyping session, and not so much during during the initial specification, because that's when the ins and outs, the really inside details that no one ever puts in a specification come to light. Until he actually sees something on the screen to play with and someone, when looking at it, will say 'But what about X?', and X here is something that no one's ever mentioned. And, that's the thing that will typically kill an application that's not been prototyped, because the unknowns that no one ever mentioned, and now you have spent some number of months developing along an avenue, and find that you've been going downtown instead of cross-town, kind of thing. And, to undo it back to the point where it's really useful to the customer is perhaps lucrative, because you can prove that he should have told you that in the first place, but it's still not what you want to be doing. It doesn't make for a successful application, because the customer will remember the wrong thing.

<u>Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?</u>

There are a couple of cases where the traditional method might do just as well, I'm not sure whether it would do better, but just as well, and that is when you have... where the person who's doing the analysis is an ex-user, from a standpoint that this person totally understands the application environment that we're dealing with. Without that, I don't think it's possible [without prototyping], for the traditional way to be better, without that in-depth knowledge.

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

If we were going to develop something without a customer [i.e., on speculation] and we wanted to go out and sell it and therefore we going to develop what we might term a complete prototype, that that prototype would have at least one of the main functions complete, in as detailed a fashion as possible [with computations coded], from input screens to file maintenance through reports, queries, the whole bit. The rest could be teasers, and might be there for one screen that doesn't go anywhere, kind of thing [i.e., stubbing employed]. But, you're not showing them [clients] anything if you're not showing them some results, what can be done here.

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

I think if you're using a tool that was designed for prototyping a system then that's [recoding for executional efficiency] probably true. However, we would not do that... Because we are going to be developing a working prototype, that is, part of that prototype is going to be functional, through computations, etc., and is also going to be using this for the basis for the end result, then we use the same tools for developing the so-called prototype as we would for the actual end application. So that the question of a prototype tool doesn't really come into play here [note respondent refers to dedicated prototyping tools; he uses Progress as his prototyping tool], as far as we're concerned, we just don't use it.

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

I think in the beginning, when we initially looked for a product, our objective was to find one that we could use for any situation that we could foresee, that could be used over a general field of machine types, so we wouldn't have to worry about hardware... And, we really looked for a tool to do the job, first, as opposed to a company. Once we got that field down to one or two or three, then we started looking at the company itself. And, in many cases in the micro[computer] world, you really couldn't look too deep, because the companies were generally new... and usually privately held and the rest of it. And, you had to use your experience in judging people more than the company itself: the type of people, the attitude. Much of it revolved around, what did the product look like? What does it look

like today? And, you can basically get a feel for the approach they have taken and when they are going. Some products, for example, might be excellent as they stand, but there's really no way that they can easily advance or enhance that product. In some cases, it's very obvious. And then you have to pick out... somewhere along the line, you have to make a decision, which may be right or wrong, and in this case we felt it was very right and that's proved to be so. The company behind Progress [respondent's prototyping tool]... is very strong. They have done a fantastic job as far as enhancements, and their approach, as to which enhancements come first, is one that I particularly like, and that is, they listen to their users, which is myself and companies like myself. That's very comforting to know that when you simply call up on the phone I think the product should be able to this or we're having a problem accomplishing something because of something which is missing in the product, to know that that is written down and discussed and this particular company has meetings with users, again meaning myself, to talk about enhancements and what is more important, and actually have an input as to what is going on.

<u>What would it take to cause you to leave your prototyping tool and move on to another?</u>

Probably, I don't know how to put that... [My] initial reaction is, an awfully lot. It gives us so many advantages that someone would have to show me something that was measurably better, no only in one way but many ways. There are many products out there today that compete with Progress, and the way I would put it is, you could take any one of them and point to some feature about that product which was better than the way Progress did it... But on an overall basis, I haven't found anything that even comes remotely close to the type of thing we want to see from a product like this. Generalized ease of development, speed of development, across machines, some mainframe capabilities such as online recovery, rollback recovery, that kind of thing.

<u>What features would you desire in a prototyping tool that are not currently available?</u>

There are so few that are not in Progress already. One could improve some of the screen painting abilities, and that is coming... to make some of that easier. The philosophy behind the way they put Progress together is to give you the you the building [tools] to do anything you want, today. How easy that is, is what the question is. So, everything is there now... [the question is] can Progress develop a tool that will make this particular function easier to do, as opposed to having the ability to do it at all.

What shortcomings do you see in the prototyping method?

I guess if I'm dealing with my definition of prototyping... the shortcoming that you tend to notice is that the initial portion of development takes longer. As I said, I still believe in the long run you wind up spending the same amount of time, but the traditional programmer does not want to deal with the user, so anytime he has to deal with someone who is not at his technical level, the typical programmer has a problem, and will view perhaps the interface with the user that I do as a pain in the neck, and it will take longer in the beginning. Somewhere or other, that program, to be successful, has to get over that hump, but the initial drawback that you seem to see is that it will take longer. A user may have a completely different idea of an approach, which in your [developer's] mind may be better or worse. And, sometimes you have to go with what the user says because you can't convince him otherwise, and he is the man with the money.

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

[Not asked.]

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

I tend to think that the user experience is invaluable. [Interviewer probes: what about experience as a developer in the application area?:] It can substitute to a great degree, really. I've seen many instances of people in my business who at least start out their going into business for themselves [by] developing a particular package... He will take something which he has been involved in typically for ten or fifteen years, some area of MIS, let's say... And, he thinks he can put together a better product... so he will open up his own business and he will write that piece of application software, and he writes it because he knows what's necessary, because he has the experience. Whether he's a user or not may not make a difference, depending on how close he has been to the product or to the other products like it, and to the operating environment, but typically he can do a fairly good job of coming up with that piece of software. But, it's questionable about what happens next for this guy's business, which is not really what you [the interviewer] want to talk about. Whether or not he can make his business a viable one after that is really up in the air, that depends on the sales of this particular product...

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

One, because I believe in the idea [of prototyping] to begin with. Two, I felt that in order to do the best job possible, I needed additional knowledge of the user environment. I know what questions to ask, I don't know what the answers are.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

I would say less because you're typically developing a series of questions in your mind, which is where the anxiety comes from, something about the particular application area which you don't know or don't understand well enough. When you're prototyping, you automatically gather those questions together and handle them in a prototype session with the user, so that you can get those questions answered up front and take the guesswork out. So, I think the anxiety over whether or not you're doing it the right way, taking the right approach, is severely cut down because of the prototyping.

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

I think it's [exercising leadership] more often because there's more contact. When you don't prototype, there are several possible results, I guess. One might result from the fact that the customer has no idea what he wants, and is willing to accept whatever you do as long as it gets the job done. The other is the case where the customer has allowed you to do this without prototyping and is not happy with the result, yet it almost does it. Your contact with the customer is now. Perhaps the programmer who is willing to have the customer spend more money to get it fixed, the way he [the customer] wants it, and perhaps in many ways you're also a diplomat trying to change the customer's mind as to what he wants...

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

Well, I don't think they [users] bring up the options on their own. They do to a certain extent. I don't know whether it's appropriate to say that the user expects you to

take the lead. I think from an applications developers standpoint that you must take the lead in drawing out those things which are important to the success of the application, which typically revolve around what the typical end user does from day to day as he tries to get this particular job done, and you've got to ask the right questions, draw them, and get all that information in front of you.

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

I have a general feeling here, at least for a small business such as mine, the traditional definition of prototyping is not truly a viable one. It may make a difference on how much money is behind the corporation to begin with. There are certain instances where I could see, you could go after seed money, for example, to develop a particular application, something which is obvious to the people who are investing that it's going to be a success, that it's needed out in the marketplace, etc.... In that case, when you have that money behind you and you're not worrying about money coming in from day to day, then you can develop the true prototyping. I feel that prototype should have some piece which is in operating order. I think the impression you make with that is so much more valuable, it's well worth the time spent. So, in that case, you could develop this prototype and use it as a marketing tool, that is, to go out and drum up business, hopefully now to get some kind of monies out of these prospective customers for you to now fund the remaining development of that project. The more [customers] you can get, the less you have to get from each one, etc. etc.. Other than that, a company such as mine, or of my size, will tend not to be able to do that, the luxury just isn't there. You must spend your time developing, and if a customer now comes to you with a particular problem, and you want to solve that problem, I think you want to get the advantages from this prototyping approach, this generalized meeting every so often, with the customer, you want to get the advantages of that, but you can't really afford to just develop the prototype and say 'Is this what you like?'. I don't want to carry it that far, because if you're off the beam to any extent, then you have just wasted hours, because you can't really charge him for that. Typically, here, you're charging him by the hour, in some fashion or other. We could charge him by strict hours spent, or, if it's a fixed price, that fixed price is developed by the hour[ly rate]... But, typically, without some large amount of monies behind you, the typical prototyping approach is not viable, you just can't do it, can't afford it. You need those monies coming in, which means you have to get started, and what you get started with is something that you're going to end up with or a piece of it... [So,] you should not be using the prototype tools that end up with the prototype and that's all, you really need something that's going to be viable and can be built upon.

EXHIBIT AE

Transcript of Responses to Selected Interview Questions

Respondent 29

Would you now share with me your own operating definition of prototyping?

It [respondent's operating definition of prototyping] matches with yours [interviewer's introductory definition].

Please think of systems that were successfully prototyped. Can you determine any common characteristics or common denominators of systems that were successfully prototyped?

No. I mean, they're all kind of look the same. A system is a system, as far as we are concerned.

Please think of systems that were not successfully prototyped, where prototyping did not work well. Can you determine any common characteristics or common denominators of systems that were not successfully prototyped?

No. Prototyping is successful, I think, almost by definition.

Let's discuss the time it takes to develop a system. Here, I refer to the calendar time from conception to delivery of a system, not man-hours of development. In general, do you think prototyping allows systems to be delivered in less calendar time than systems developed in the traditional manner? Why?

Because if you don't prototype, essentially, you have to do it at least twice, or maybe more. Even if you prototype you may have to do it twice, but at least then you're doing little chunks at a time, and you don't go off on tangents.

In general, do you think prototyping allows systems to be developed at less cost than systems developed in the traditional manner? Why?

It shortens the time frame [meaning] less cost.

In general, do you think systems that are prototyped are of lesser quality, equal quality or greater quality than those developed using more traditional approaches? Why?

[Not asked.]

Some people feel prototyping is an effective method of understanding or eliciting user needs. Others feel that more traditional analysis and design methods are a more effective way of understanding user needs. What do you think?

Well, generally... it depends on the level of people that you're having doing the systems design. If they have low level people doing it [systems design] they have no choice but go through the flowcharting and everything else. But, if they have systems people working it, rather than programmers, it's a different story...

Some people feel that prototypes should consist of a simulation of the proposed system, with no actual underlying computations coded. Others feel actual computations should be included in the prototype. What do you think?

It's just as easy to do it with the actual [computations] so you might as well use the actual [computations]. I don't think it makes an difference, but why use simulated data when you can use the actual data?...

Say we employ an evolutionary approach [i.e., the prototype evolves into the production system]. Some critics argue that an evolved system is executionally inefficient, and that recoding in a lower level language is necessary for performance reasons. What do you think?

...It's our feeling that the machines are getting so fast now that while at one point that [refers to rewriting for executional efficiency] was a pretty valid criticism, I think it's quickly disappearing.

How do you judge the stability and the longevity of the firm from which you will purchase your prototyping tool(s)?

Educated guess. We put our whole database in 1980 on Progress and they hadn't sold a copy. They had zero revenue, they were just on the drawing boards. But, we analyzed it from a technical point of view and to it seemed like it couldn't fail. It was exactly what we were looking for and there was no one else doing it; it was probably the biggest risk we ever took.

What would it take to cause you to leave your prototyping tool and move on to another?

Significant improvements. At least a factor of 100%, something like that.

What features would you desire in a prototyping tool that are not currently available?

Well, there's some pretty technical things in Progress [respondent's prototyping tool] that we'd like to have changed. There's not many, but there are some. [Interviewer probes: what specific features are desired?:] Well, they're pretty technical features: we'd like a better report writer, a real good report writer. I'm not sure we can get one, but that's what we'd like. We'd like to have a little more portability of the database itself; in other words, right now, it's a little cumbersome, even though it's very portable, it's a little cumbersome to make actual database changes, not for us but for the end user. I think of fairly technical things such as that. They're [Progress] working on that...

What shortcomings do you see in the prototyping method?

As opposed to other [systems development] methods, I don't see any [shortcomings to prototyping]. There are certain shortcomings, but it's better than any other method, it's just not the ultimate. [Interviewer probes:] The fact that every time you change an element in the database, you have to actually recompile any program that are associated with that. And we carry prototyping to an extreme, we prototype entire systems and when me make a change to any component of the system, right now if that involves a database change, you have to recompile the whole system. That would be one major feature that we'd like not to have to do.

This question deals with the relationship of prototyping and your operations as a business. I want to assure you that I am not requesting privileged information. What relationship do you see between prototyping and the operations of your business, particularly the financial operations?

[Not asked.]

Say that a developer has very substantial experience with a system similar to that proposed. To what degree can that experience effectively substitute for prototyping?

It can substitute a lot. It can't replace it.

[Addressing a specific prototyping experience:] Why did you consciously choose to prototype this project?

Because we prototype all of our projects. That's just the way we do business.

When you are involved in prototyping, how do you approach the project? What is your plan of attack?

[Not transcribed.]

All developers feel some anxiety when approaching a new project. When prototyping, do you feel more anxious or less anxious than when you employ traditional development?

Less anxious because you get feedback much earlier on in the cycle.

When prototyping, do you find you must take a leadership role with respect to clients more often or less often compared to traditional development?

[Not transcribed.]

In general, when prototyping, do users want you to take the lead in present various system options or do they bring up sufficient options on their own?

It all depends on the user but, generally, no [the developer must present system options].

I would like to leave the tape recorder running and gather any other thoughts that you would like to share with me.

Well, it [prototyping] works. What you're [to interviewer] doing is putting more emphasis on prototyping. I think there are more people [systems developers] doing it than you think... When we prototype, that's a crucial thing for us to do. The thing that's of more concern to us is the 4GL [fourth generation language]... We always come at a project as if we don't know anything about it, and the people who know something about it are the people we are trying to serve, and now we're forced to communicate with them, and that's the only way I'd be able to do it... We wouldn't be a business without it [their fourth generation language, Progress]... We only have about twenty-five people, but they're twenty-five good people, and with the prototyping language you can, I'd say, ten times the work out of a good person than you can [get] out of an ordinary person... you can do a lot of work if you use the tools properly...

BIBLIOGRAPHY

Ackoff, Russell L., "Management misinformation systems", Management Science 14:4 (December 1967): B-147 - B-156

Alavi, Maryam, "An assessment of the prototyping approach to information systems development", Communications of the ACM 27:6 (June 1984): 556-563

Alavi, Maryam, and John C. Henderson, "An evolutionary strategy for implementing a decision support system", Management Science 27:11 (November 1981): 1309-1323

Albano, Antonio and Renzo Orsini, "A prototyping approach to database applications development", Database Engineering 7:4 (December 1984): 64-69

Andrews, William C., "Prototyping information systems", Journal of Systems Management 34:9 (September 1983): 16-18

Association for Systems Management and Tor Guimaraes, Prototyping implementation strategies, Cleveland: the Association, 1986

Bally, Laurent, John Brittan, and Karl H. Wagner, "A prototype approach to information systems development", Information & Management 1 (1977): 21-26

Berrisford, Thomas and James C. Wetherbe, "Heuristic development: a redesign of systems design", MIS Quarterly 3:1 (March 1979): 11-19

Blum, Bruce I., "The life cycle - a debate over alternate models", Software Engineering Notes 7:4 (October 1982); 18-20

_____, "Four yrs [sic] experience with an environment for implementing information systems", Journal of Systems and Software 6:3 (August 1986): 261-271

_____, "Iterative development of information systems: a case study", Software - Practice and Experience 16:6 (June 1986): 503-515

Boar, Bernard, Application prototyping: a requirements definition strategry for the 80's, New York: John Wiley & Sons, 1984

Boehm, Barry W., "Software and its impact: a quantitative assessment", Datamation (May 1973): 48-59

_____, "Software engineering", IEEE Transactions on Computing C-25 (December 1976): 1226-1241

_____, Software engineering economics, Englewood Cliffs, NJ: Prentice-Hall, 1981

_____, [Discussions on] "Prototyping versus specifying: a multiproject experiment", pp. 146-147 in Proceedings of software process workshop, Egham, Surrey, UK, 6-8 February 1984, ed. by Colin Potts, Silver Spring, MD: IEEE Computer Society Press, 1984

_____, Terence E. Gray and Thomas Seewaldt, "Prototyping versus specifying: a multiproject experiment", IEEE Transactions on Software Engineering SE-10:3 (May 1984): 290-302

Boston Computer Society, The BCS computers, software and services buyers guide, Boston: The Society, June-November 1987

Bottom, Joseph, Alan Bernard and Kevin Anderson, "The art of modeling", Datamation 31:22 (November 15, 1985): 140-146

Bricklin, Dan, Dan Bricklin's Demo Program, Cambridge, MA: Software Garden, 1985

Brooks, Frederick P., Jr., The mythical man-month: essays on software engineering, Reading, MA: Addison-Wesley, 1975

Burns, A. and J.A. Kirkham, "The construction of information management system prototypes in Ada", Software - Practice and Experience 16:4 (April 1986): 341-350

Carey, T.T. and R.E.A. Mason, "Information system prototyping: techniques, tools and methdologies", INFOR 21:3 (August 1983): 177-191

Carper, William B., "Human factors in MIS", Journal of Systems Management 28:11 (November 1977): 48-50

Cerveny, Robert P., and Thomas D. Clark, Jr., "Conversations on Why information systems fail - and what can be done about it", Systems, Objectives, Solutions 1 (1981): 149-154

_____, Edward J. Garrity and G. Lawrence Sanders, "The application of prototyping to systems development: a rationale and model", Journal of Management Information Systems 3:2 (Fall 1986): 52-62

_____, Edward J. Garrity, Raymond G. Hunt, Peter J. Kirs, G. Lawrence Sanders and Janice C. Sipior, "Why software prototyping works", Datamation 33:16 (August 15, 1987): 97-103

Chorafas, Dimitris, Designing and implementing local area networks, New York: McGraw-Hill Book Company, 1984

Cohen, I. K., and Richard L. Van Horn, A laboratory research approach to organizational design, Rand Corporation Report P-4776, February 1972

Connell, John L., and Linda Brice, "The impact of implementing a rapid prototype on system maintenance", pp. 515-524 in Proceedings, 1983 national computer conference, Anaheim, 16-19 May 1983, Arlington, VA: AFIPS Press, 1983

Cotterman, William W., J. Daniel Cougar, Norman L. Enger and Frederick Harold, eds., Systems analysis and design: a foundation for the 1980's, Amsterdam: North-Holland, 1981

Cougar, J. Daniel, "Evolution of business systems analysis techniques" Computing Surveys 5:3 (September 1973): 167-198

Dagwell, Ron, and Ron Weber, "Systems designers' user models: a comparative study and methodological critique", Communications of the ACM 26:11 (November 1983): 987-997

Data Language Corporation, Progress application catalog, Billerica, MA: Data Language Corporation, 1986

Davis, Gordon B., Management information systems: conceptual foundations, structure, and development, New York: McGraw-Hill, 1974

_____, "Strategies for information requirements determination", IBM Systems Journal 21:1 (January-March 1982): 4-30

_____, and Margrethe H. Olson, Management information systems: conceptual foundations, structure, and development, 2nd ed., New York: McGraw-Hill, 1985

Davis, K. Roscoe, and Bernard W. Taylor, "Systems design through gaming", Journal of Systems Management 26:9 (September 1975): 36-42

De, Prabuddha, and Cheng Hsu, "Information systems evolution through users' feedback", pp. 661-669 in Proceedings of the eighteenth Hawaii international conference on system sciences, 1985, volume 1, the Conference, 1985.

DeGross, Janice I., and Charles H. Kriebel, eds., Proceedings of the eighth international conference on information systems, December 6-9, 1987, Pittsburgh, Pennsylvania

DeMarco, Tom, Controlling software projects: management, measurement & estimation, New York: Yourdon Press, 1982

DeSanctis, Gerardine, and James F. Courtney, "Toward friendly user MIS implementation", Communications of the ACM 26:10 (October 1983): 732-738

Earl, Michael J., "Prototype systems for accounting, information and control", Accounting, Organizations and Society 3:2 (September 1978): 161-170

_____, "Prototype systems for accounting, information and control", Data base 13:2-3 (Winter-Spring 1982): 39-46

Edelman, Franz, "Managers, computer systems and productivity", MIS Quarterly 5:3 (September 1981): 1-19

Edström, Anders, "User influence and the success of MIS projects: a contingency approach", Human Relations 30:7 (1977): 589-607

Proceedings of the eighteenth Hawaii conference on system sciences, 3 vols., the Conference, 1985

Proceedings, 5th [sic] international conference on software engineering, San Diego, CA, March 9-12, 1981, Los Alamitos, CA: IEEE Computer Society Press, 1981

Fok, Lillian Yee-man, Kuldeep Kumar and Trevor Wood-Harper, "Methodologies for socio-technical systems development", pp.319-334 in Proceedings, Eighth Annual Conference on Information Systems, Pittsburgh, December 6-9, 1987, ed. by Janice I. DeGross and Charles H. Kriebel

Frank, James W., "Application design by trial and error", Infosystems, 26:9 (September 1979): 76-78

Gane, Chris, and Trish Sarson, Structured systems analysis: tools and techniques, Englewood Cliffs, NJ: Prentice-Hall, 1979

Gibson, Harry L., "Determining user involvement", Journal of Systems Management 28:8 (August 1977): 20-22

Giddings, Richard V., "Accommodating uncertainty in software design", Communications of the ACM 27:5 (May 1984): 428-434

Ginzberg, Michael J., "Early diagnosis of MIS failure: promising results and unanswered questions", Management Science 27:4 (April 1981): 459-478

_____, Walter Reitman and Edward A. Stohr, eds., Decision support systems: proceedings of the NYU Symposium on decision support systems, New York, 21-22 May, 1981, Amsterdam: North-Holland Publishing, 1982

Gomaa, Hassan, and Douglas B.H. Scott, "Prototyping as a tool in the specification of user requirements", pp. 333-339 in Proceedings, 5th [sic] international conference on software engineering, San Diego, CA, March 9-12, 1981, Los Alamitos, CA: IEEE Computer Society Press, 1981

Good, Michael D., John A. Whiteside, Dennis R. Wixon and Sandra J. Jones, "Buidling a user-derived interface", Communications of the ACM 27:10 (October 1984): 1032-1043

Goodman, A.M., "IMSADF: a tool for programmer productivity", Data base 11:3 (Winter-Spring 1980): 106-113

Gore, Marvin, and John Stubbe, Elements of systems analysis, 3rd ed., Dubuque, Iowa: Wm. C. Brown Publishers, 1983

Guimaraes, Tor, "Managing application program maintenance expenditures", Communications of the ACM 26:10 (October 1983): 739-745

_____, "A study of application program development techniques", Communications of the ACM 28:5 (May 1985): 494-499

_____, "Prototyping: orchestrating for success", Datamation 33:23 (December 1, 1987): 101-106

Hanson, Stephen Jose, and Richard R. Rosinski, "Programmer perceptions of productivity and programming tools", Communications of the ACM 28:2 (February 1985): 180-189

Hare, Van Court, Jr., Systems analysis: a diagnostic approach, New York: Harcourt, Brace & World, 1967

Harel, David, "On folk theorems", Communications of the ACM 23:7 (July 1980): 379-389

Hartson, H. Rex, ed., Advances in human-computer interaction, vol. 1, Norwood, NJ: Ablex Publishing, 1985

Hayes, Robert H., and Richard L. Nolan, "What kind of corporate modeling functions best?", Harvard Business Review 52:3 (May-June 1974): 102-112

Hedberg, Bo, and Enid Mumford, "The design of computer systems: man's vision of man as an integral part of the system design process", pp. 31-59 in Human choice and computers: proceedings of the IFIP conference on human choice and computers, Vienna, April 1-5, 1974, ed. by Enid Mumford and Harold Sackman, Amsterdam: North-Holland Publishing, 1975

Henderson, John C., and Robert S. Ingraham, "Prototyping for DSS: a critical appraisal", pp. 79-96 in Decision support systems: proceedings of the NYU symposium on decision support systems, New York, 21-22 May, 1981, ed. by Michael J. Ginzberg et al, Amsterdam: North-Holland Publishing, 1982

Holmes, Fenwicke W., "The many roles of the user in systems development", Data base 9:4 (Spring 1978): 19-21. This article was a section in the "Miscellany" column.

Ives, Blake, Scott Hamilton and Gordon B. Davis, "A framework for research in computer-based management information systems", Management Science 26:9 (September 1980): 910-934

_____, and Gerard P. Learmonth, "The information system as a competitive weapon", Communications of the ACM 27:12 (December 1984): 1193-1201

_____, and Margrethe H. Olson, "User involvement and MIS success: a review of research", Management Science 30:5 (May 1984): 586-603

Janson, Marius A., and L. Douglas Smith, "Prototyping for systems development: a critical appraisal", MIS Quarterly 9:4 (December 1985): 305-316]

Jenkins, A. Milton, Justus D. Naumann and James C. Wetherbe, "Empirical investigation of systems development practices and results", Information & Management 7 (1984): 73-82

Johnson, James R., "A prototypical success story", Datamation 29:11 (November 1983): 251-256

Jung, C.G., Psychological types; volume 6 of the collected works of C.G. Jung (Bollingen Series XX), edited by Sir Herbert Read, Michael Fordham, Gerhard Adler and William McGuire, Princeton, NJ: Princeton University Press, 1971

Keen, Peter G. W., "Information systems and organizational change", Communications of the ACM 24:1 (January 1981): 24-33

Keen, Peter G. W., and Elihu M. Gerson, "The politics of software systems design", Datamation 23:11 (November 1977): 80-84

King, William R., "Alternative designs in information system development", MIS Quarterly 6:4 (December 1982): 31-42

_____, and Jaime I. Rodriguez, "Evaluating management information systems", MIS Quarterly 2:3 (September 1978) 43-51

_____, and David I. Cleland, "Manager-analyst teamwork in MIS: cooperation vital in systems design", Business horizons 14:1 (April 1971): 59-68

Kling, Rob, and Suzanne Iacono, "The control of information systems developments after implementation", Communications of the ACM 27:12 (December 1984): 1218-1226

_____, "The organizational context of user-centered software designs", MIS Quarterly 1:4 (December 1977): 41-52

Kneitel, Arnold M., "Fables for systems designers", Journal of Systems Management 28:10 (October 1977): 9-11

Kolb, David A., I.M. Rubin and J.M. McIntyre, Organizational psychology: an experimental approach, third ed., Englewood Cliffs, NJ: Prentice-Hall, 1979

_____, and Alan L. Frohman, "An organizational development approach to consulting", Sloan Management Review 12:1 (1970): 51-65

Kraushaar, James M., and Larry E. Shirland, "A prototyping method for applications development by end users and information systems specialists", MIS Quarterly 9:3 (September 1985): 189-197

Kruchten, Phillippe, Edmond Schonberg and Jacob Schwartz, "Software prototyping using the SETL programming language", IEEE Software (October 1984): 66-75

Kuhn, Thomas S., The structure of scientific revolutions, Chicago: University of Chicago Press, 1982

Lantz, Kenneth E., The prototyping methodology, Englewood Cliffs, NJ: Prentice-Hall, [1987]

Lehman, Meir M., "Programs, life cycles and the laws of software evolution", Proceedings of the IEEE 69:9 (September 1980): 1060-1076

Lehman, Meir M., "A further model of coherent programmaning processes", pp. 27-35 in Proceedings of software process workshop, Egham, Surrey, UK, 6-8 February 1984 (Colin Potts, ed.), Silver Spring, MD: IEEE Computer Society Press, 1984

Lewin, K., "Group decision and social change", pp. 330-344 in Readings in Social Psychology, ed. by T.M. Newcomb and E.L. Hartley, New York: Holt, 1947

Lientz, Bannet P. and E. Burton Swanson, Software maintenance management: a study of the maintenance of computer application software in 487 data processing organizations, Reading, MA: Addison-Wesley Publishing Co, 1980.

Lientz, Bannet P. and E. Burton Swanson, "Impact of development productivity aids on application system maintenance", Data base 11:3 (Winter-Spring, 1980): 114-120

Lirov, Yuval and Nissim Daunov, "Heuristic approach to network database external parameters design", Information Systems 10:4 (1985): 371-376

Livingston, Dennis, "Software links multivendor networks" Mini-Micro Systems 21:3 (March, 1988) 43-54

Lucas, Henry C., Jr., "A user-oriented approach to system design", pp. 325-338 in Proceedings, 1971 national computer conference, New York: Association for Computing Machinery, 1971

Lucas, Henry C., Jr., Why information systems fail, New York: Columbia University Press, 1975

Mahmood, Mo A., "Systems development nethods - a comparative investigation", MIS Quarterly 11:3 (September 1987): 293-311

Maish, Alexander M., "A user's behavior toward his MIS", MIS Quarterly 3:1 (March 1979): 39-52

Markus, M. Lynne, "Power, politics and MIS implementation", Communications of the ACM 26:6 (June 1983): 430-444

Martin, James, with research by Richard Murch, Application development without programmers, Englewood Cliffs, NJ: Prentice-Hall, 1982

Mason, R.E.A., and T.T. Carey, "Prototyping interactive information systems", Communications of the ACM 26:5 (May 1983): 347-354

Mason, Richard O., and Ian I. Mitroff, "A program for research on managment information systems", Management Science 19:5 (January 1973): 475-487

McCracken, Daniel D., and Michael A. Jackson, "A minority dissenting position", pp. 551-553 in Systems analysis and design: a foundation for the 1980's, ed. by William W. Cotterman et al, Amsterdam: North-Holland, 1981.

McLean, Ephriam R., "The concept of throwaway code", Datamation 23:3 (March 1977): 139-144

McMillan, Claude, and Richard F. Gonzalez, Systems analysis: a computer approach to decision models, 3rd ed., Homewood, IL: Richard D. Irwin, 1973

McNurlin, Barbara Canning, "Developing systems by prototyping", EDP Analyzer 19:9 (September 1981): 1-14

Meredeth, Denis C., "Systems development methodologies provide project fire power", Data Management (August 1985): 22-24

Michielsen, Ken, "Micro applications development", Datamation 32:8 (April 15, 1986): 96-98

Microrim, Inc., Microrim applied: a guide to over 200 applications and over 200 Microrim authorized consultants, Bellevue, WA: Microrim, Inc., 1986

Mitroff, Ian I., Frederick Betz, Louis R. Pondy and Francisco Sagasti, "On managing science in the systems age: two schemas for the study of science as a whole systems phenomenon", Interfaces 4:3 (May 1974): 46-58

Mumford, Enid, and Harold Sackman, eds., Human choice and computers: proceedings of the IFIP conference on human choice and computers, Vienna, April 1-5, 1974, Amsterdam: North-Holland Publishing, 1975

Murdick, Robert G., MIS concepts and design, Englewood Cliffs, NJ: Prentice-Hall, 1980

Naumann, Justus D., and A. Milton Jenkins, "Prototyping: the new paradigm for systems development", MIS Quarterly 6:3 (September 1982): 29-44

Necco, Charles R., Carl L. Gordon and Nancy W. Tsai, "Systems analysis and design: current practices", MIS Quarterly 11:4 (December 1987): 461-476

Proceedings, 1971 national computer conference, New York: Association for Computing Machinery, 1971

Proceedings, 1983 National Computer Conference, Anaheim, 16-19 May 1983, Arlington, VA: AFIPS Press, 1983

Olson, Margrethe H., and Blake Ives, "User involvement in system design: an empirical test of alternative approaches", Information & Management 4 (1981): 183-195

Potts, Colin, ed., Proceedings of software process workshop, Egham, Surrey, UK, 6-8 February 1984, Silver Spring, MD: IEEE Computer Society Press, 1984

Powers, Richard F., and Gary W. Dickson, "MisProject [sic] management: myths, opinions and reality", California Management Review, 15:3 (Spring, 1973) 147-156

Riddle, William E., Jack C. Wileden, John H. Sayler, Alan R. Segal, and Allan W. Stavely, "Behavior modelling [sic] during software design", pp. 13-22 in Proceedings, 3rd international conference on software engineering, May 10-12, 1978, Atlanta, GA, New York: IEEE Computer Society, 1978

Salaway, Gail, "An organizational learning approach to information systems development", MIS Quarterly 11:2 (June 1987): 245-264

Sarvari, I.L., "The case for prototyping in systems development", Canadian Datasystems (October 1983): 100-104

Sauter, Vicki L., and Joseph L. Schofer, "Evolutionary development of decision support systems: early phases of design", Journal of Management Information Systems 4:4 (Spring 1988): 77-92

Schewe, Charles D., and James L. Wiek, "Guide to MIS user satisfaction", Journal of Systems Management 28:6 (June 1977): 6-9

Scott, James H., "The management science opportunity: a systems development management viewpoint", MIS Quarterly 2:4 (December 1978): 59-61

Senn, James A., Analysis and design of information systems, New York: McGraw-Hill , 1984

Simkin, Mark G., Introduction to computer information systems, Dubuque, Iowa: Wm. C. Brown Publishers, 1987

Smith, C. Peter, "Resolving user/systems difference", Journal of Systems Management 28:7 (July 1977): 16-21

Stemple, David, Tim Sheard, and Ralph Bunker, Incorporating theory into database system development, COINS Technical Report 85-08, Computer and Information Science Department, University of Massachusetts at Amherst, March 1985

Swanson, E. Burton, "Management information systems: appreciation and involvement", Management Science 21:2 (October 1974): 178-188

_____, Information systems implementation, Homewood, IL: Irwin, 1988

Tersine, Richard J., and Walter E. Riggs, "Models: decision tools for management", Journal of Systems Management 27:10 (October 1976): 30-34

Proceedings, 3rd international conference on software engineering, May 10-12, 1978, Atlanta, GA, New York: IEEE Computer Society, 1978

Urban, Glen L., and Richard Karash, "Evolutionary model building", Journal of Marketing Research VIII (February 1971): 62-66

Wasserman, Anthony I., "User software engineering and the design of interactive systems", pp. 387-393 in Proceedings, 5th international conference on software engineering, San Diego, CA, March 9-12, 1981, Los Alamitos, CA: IEEE Computer Society Press, 1981

Wasserman, Anthony I., "Characteristics of the user software engineering methodology", pp. 125-125 in Proceeding of software process workshop, Egham, Surrey, UK, 6-8 February 1984, Colin Potts, ed., Silver Spring, MD: IEEE Computer Society Press, 1984

Wasserman, Anthony I., and David T. Shewmake, "The role of prototypes in the User Software Engineering (USE) Methodology", pp. 191-209 in Advances in human-computer interaction, vol. 1 (H. Rex Hartson, ed.,), Norwood, NJ: Ablex Publishing, 1985

Waters, S. J., "Towards comprehensive specifications", The Computer Journal 22:5 (August 1979): 195-199

Weisman, Randy, "Six steps to AI-based functional prototyping", Datamation 33:15 (August 1, 1987): 71-72

Wetherbe, James C., "Advanced system development techniques avoid 'analysis paralysis'", Data Management 22:2 (February 1984): 49-51

Wetherbe, James C., and Thomas R. Berrisford, "Narrowing the expectations gap", systems/stelsels 9:8 (August 1979): 10-14. This piece was a presentation given at the 7th International Adabas Users Group meeting, San Diego, CA, 1979

Whitten, Jeffrey L., Lonnie D. Bentley and Thomas I.M. Ho, Systems analysis & design methods, St. Louis: Times Mirror/Mosby, 1986

Young, T.R., "Superior prototypes", Datamation 30:7 (May 15, 1984): 152-156

Yourdon, Edward, Techniques of program structure and design, Englewood Cliffs, NJ: Prentice-Hall, 1975

Yourdon, Edward, "What ever happened to structured analysis?", Datamation 32:11 (June 1, 1986): 133-138

Zave, Pamela, "The operational approach versus the conventional approach to software development", Communications of the ACM 27:2 (February 1984): 104-118

Zelkowitz, Marvin V., "A case study in rapid prototyping", Software - Practice and Experience 10:12 (December 1980): 1037-1042